LAYOUT INFERENCE:

FILE SCHEMA RECOGNITION VIA CONTENT-BASED ORACLES

LAYOUT INFERENCE:

FILE SCHEMA RECOGNITION VIA CONTENT-BASED ORACLES

A dissertation submitted in partial
fulfillment of the requirements for the degree of
Doctor of Philosophy

By

Reid A. Phillips
University of Arkansas, 2006
Master of Science in Computer Science

August 2009
University of Arkansas

ABSTRACT

Some organizations routinely (e.g., monthly) process tens of thousands of flat files, files consisting of records containing a fixed number of fields, received from third parties. Currently, the process of characterizing each file's encoding, formatting elements, structure, and content is a manual process, expensive in that the process costs human time, delays processing the files, and is error prone. This dissertation provides methods for automatically inferring the specified meta data associated with these files.

In order to mine, persist, transform, or in some other way process structured data contained within a flat file, the properties associated with a file must first be known. Within this paper, the identification of these properties will be referred to as the *layout inference problem*, where a layout is a specification of the characteristics associated with a file. Typically a manual task, layout inference can benefit from an automated tool designed to replace or assist human involvement in this process.

In defining the result of this process, the layout, the first step is to identify the properties to be inferred. These characteristics are requisite to read and process the contents of a file and include but are not necessarily limited to: the schema of the data records contained within a file, the character encoding, and other formatting details. Thus layout inference is concerned with providing an encompassing description of a file rather than a single characteristic (e.g., only the character encoding). Once available, the final step in the layout inference problem is to communicate the produced layout in a meaningful manner to any interested parties.

The approach to this problem described in this paper is primarily statistical in nature. Statistical solutions, while potentially more ambiguous, can be considered to be better than other solutions because they are more adaptive: gracefully handling a limited amount of error and incomplete information along with many unforeseen circumstances. Another important characteristic of the approach detailed herein is a conglomeration of expert agents. These agents provide the means for identification of the file properties as each agent is an expert concerning a respective property. By applying their respective knowledge in various ways, as appropriate with respect to the property being determined, the various layout characteristics may be inferred. Together the statistical results of expert knowledge agents provide a powerful approach to solving the layout inference problem.

The applicability of this approach towards the layout inference problem will be shown through results generated by an implemented prototype. These results will indicate the prototype's performance (i.e., accuracy and run time) with respect to a representative set of data files; consequently showing the ability of the defined approach and the promise related to certain areas of future work.

This thesis is approved for recommendation to the Graduate Council.

Thesis Directors:

_____
Wing-Ning Li

_____
Craig Thompson

Thesis Committee:

_____
Gordon Beavers

_____
David Douglas

THESIS DUPLICATION RELEASE

I hereby authorize the University of Arkansas Libraries to duplicate this thesis when needed for research and/or scholarship.

Agreed

_____

Refused

_____

ACKNOWLEDGEMENTS

I thank the Lord, Jesus Christ, for granting me the ability and endurance necessary for this project and my educational career.

I am grateful to my parents, Timothy and Charmaine Phillips, for their continuous support. In all aspects of my life their assistance and guidance has been a wonderful blessing.

I thank my advisors, Drs. Craig Thompson and Wing-Ning Li, for their advice and direction concerning both this thesis and graduate research. It is unlikely I would have reached this point without their continual patience, guidance, and support. I also appreciate Drs. Gordon Beavers and David Douglas: Dr. Beavers for his involvement in this research effort and both for their assistance as committee members.

I also thank Acxiom Corporation for sponsoring me in present and past research projects, especially for suggesting the Layout Inference problem and providing guidance and test data. I especially thank Jonathan Loghry and David Nash for sharing their understanding of the problem and reviewing this work.

Finally, I would like to also thank Wesley Deneke and Patrick Benham who have both directly contributed to this project. Your assistance has had a significant impact. Other friends, who have contributed in a less direct, yet no less meaningful way, include: Evan Kirkconnel, Jonathan Fleming, Kyle Stacey, Adam Vermillion, Nick LaSorte, Jonathan Schisler, and John Dixon. I appreciate you all.

My thesis is dedicated to all of you.

TABLE OF CONTENTS

LIST OF FIGURES

1. INTRODUCTION

1.1 Problem Definition

A common practice when transimitting data stored in a database is to pull the data and save it to a flat file in order to facilitate the transfer. In this situation the records stored in the file mirror those existing in the database. While this procedure is common to many organizations on a smaller scale, some organizations can receive up to tens of thousands of files from third parties each month for processing (e.g., customer lists). Often, the format of these files is unknown, partially known, or miscommunicated and as a result humans must examine each file manually in order to identify and specify a format. As proper extraction of the data contained within the file is the desired result, this step must be perfomed before any further processing can be performed. Even if a format is pre-specified, it is not uncommon for these format definitions to be incorrect or, in the case of monthly transmissions of the same file, to change unexpectedly. For many organizations, but especially those that aggregate data from dozens or even hundreds of data sets, identifying the layout of incoming data files is a time critical task. Considering this, and also that the job of characterizing files is expensive with respect to time, human intensive, and error prone; an automated program that can flexibly read files and assign layouts would be very beneficial. Even if standards specifying the transfer of data were to be developed and improved, many organizations may not use them causing the problem of determining file layouts to endure.

As suggested, the layout inference problem is the process of determining previously unknown structural and formatting properties of a structured (i.e., schema-based) data file from an existing set of potential attributes and reporting those

characteristics: converting an undefined file into a well-defined file. For this problem, a structured data file is a file containing consistently formatted, sequential records of data. The structure and data descriptors of the file are a consequence of the records contained within the file. Furthermore, the records are built from fields of data. Each field is defined by a positional element and a type label. Beyond these structural components, there are certain formatting elements associated with a file whose recognition is also part of the layout inference problem; specifically the character encoding and delimiters. Distinct from the record data, these items are nonetheless important, enabling data to be correctly read and extracted from a file. From the perspective of this problem, recognition of these elements is what differentiates an undefined file from a well-defined file.

Where general structured files can contain repeated subsequences of records, flat files are a restricted subset of structured files which contains only fields and not hierarchies of repeated groups of fields. The layout inference problem is restricted to flat files. Often, files of this type are produced when entities exchange data between databases. In this situation, the records contained within a database are dumped to a flat file which retains the existing structure. Once the file is created, it is transmitted to another party where the records must be parsed from the file before being processed. When the layout is unknown or miscommunicated, the receiving party must perform layout inference.

Given a flat file as input, often pulled from some database, the layout inference problem seeks to identify certain structural and formatting elements and reports them in an understandable manner. The formatting properties include the character encoding and

delimiters. The structural properties include the records and fields along with their corresponding descriptive elements: length for records and position and type label for fields. Identification is constrained to a predefined list of possible values for each property (e.g., only certain delimiters and only certain field types are considered). While each constraint may be expanded such that the supporting lists handle additional eventualities, this problem assumes the properties to be inferred are predefined. Once the layout has been inferred, the final step is to output the report in a manner understandable within the context it is used. This means that it must be both easily readable (e.g., formatted text for humans or a well structured document such as XML for machine processing) and also must contain enough information such that it is possible to ascertain how the results were derived (i.e., an inclusion of the evidence used to infer the layout properties). From input to output, this provides an introductory definition of the layout inference problem discussed in this paper.

The layout inference problem is not appropriate for all types of data. Files of free text, those containing well-formatted structural elements such as XML tags, and binary data are examples of incongruous data. Layout inference is unsuitable for free text as it relies on consistency within the data and also because the approach taken for tokenization is necessarily different since whitespace and punctuation, the normal characters used for tokenization, play a different role for this problem. This puts it at direct odds with the problem of identifying information within free text. The problem of correctly representing files containing structural elements such as XML tags is also different than this problem. While there can be consistency in such a file, it is not guaranteed; also tokenization is simplified through the existence of the formatting tags and layout

inference is unnecessary. The last example, binary data, is also not associated with the layout inference problem. While it is possible that common types of binary data such as multimedia could be recognized, the many proprietary types and forms binary data can assume make it unlikely that a general solution might be achieved. After a brief description of what the layout inference problem is, these examples are representative of what layout inference is not.

Solving the layout inference problem is typically a highly manual task as no known automated solution exists. Often, existing tools will pull a sample of data allowing a human user to identify any file properties. As this can be a mundane process, requiring unnecessary amounts of time, the intent of this research effort is to describe the means by which to automate a solution – see Figure 1. This solution must be able to identify various attributes normally associated with a layout and identified by a human: properties that make the data within the file accessible to other processes. This is important as layout inference is rarely an end in itself, often serving as the beginning of a sequence of processes which may include, among others: data hygiene, data mining, or an extract, transform, and load (ETL) sequence [2].

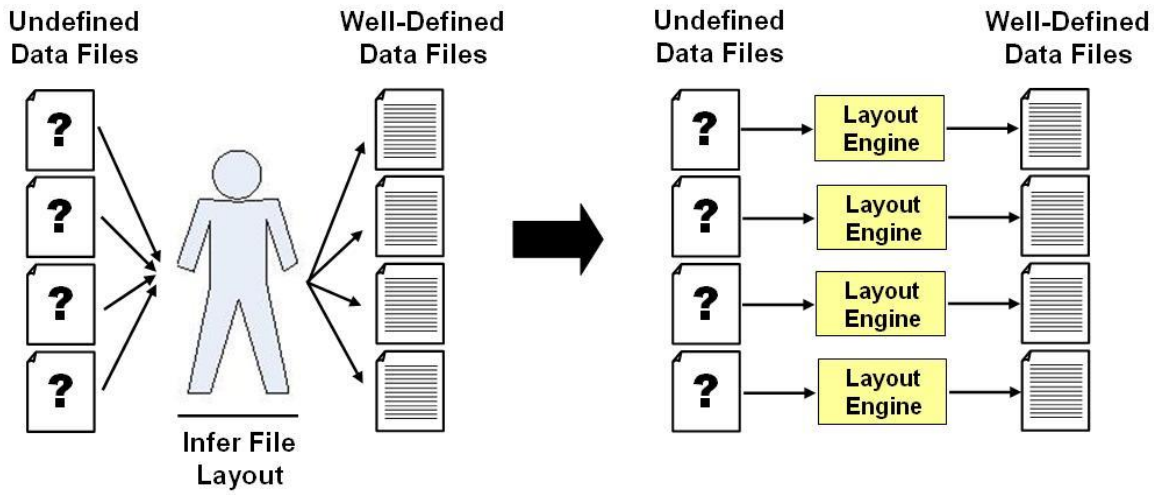

Figure 1: Visual depiction of the layout inference process: both manual (left) and automated (right).

For the described problem, three common types of files are considered, files which will be designated as fully delimited, fully fixed, and hybrid. As the terms indicate, the types of delimiters present, or not, in the file determine the type of file. A delimiter is a character, or group of characters acting as a unit, that designate boundaries between data entities. Understanding the different file formats associated with this problem is important as it affects how the layout is inferred.

Fully delimited files contain delimiters separating the fields, field delimiters, within the records and also the records themselves, record delimiters. Figure 2 depicts a fully delimited file as it would be seen in a basic text editor with commas as field delimiters and the carriage return, line feed combination as record delimiters.

```
Kenneth Mitchell,5547 East Eighth Court,Apt. 70,KITTANNING,PA,16201
Elizabeth Thomson,7944 West Washington Court,Apt. 44,BATAVIA,IL,60510
Zachary Campbell,7982 East Taft Court,,DINUBA,CA,93618
Steven Davis, 3038 West Bush Avenue,Apt. 81,DEXTER,OR,97431
Jacob Taylor,419 South Calhoun Street,,LANCASTER,WI,53813
Ryan Paterson,2888 East Carter Street,Apt. 2,INDIANAPOLIS,IN,46260
David Paterson,651 South Kennedy Court,,EL PASO,TX,79902
James Mitchell,5005 North Bear Avenue,Apt. 92,COPPELL,TX,75019
William Clark,4057 West Kennedy Expressway,Apt. 16,CONSHOHOCKEN,PA,19428
Jessica Thompson,5788 East Tenth Court,,CROSS JUNCTION,VA,22625
Anna Smith,8251 West Oak Boulevard,,ROME,NY,13440
```

Figure 2: A fully delimited file as viewed in a text editor.

```
Kenneth Mitchell           5547 East Eighth Court     Apt. 70                    KITTANNING
PA 16201Elizabeth Thomson          7944 West Washington Court Apt. 44                    BATAVIA
        IL 60510Zachary Campbell           7982 East Taft Court
DINUBA            CA 93618Steven Davis               3038 West Bush Avenue      Apt. 81
        DEXTER            OR 97431Jacob Taylor              419 South Calhoun Street
                LANCASTER         WI 53813Ryan Paterson             2888 East Carter Street
Apt. 2                    INDIANAPOLIS      IN 46260David Paterson             651 South Kennedy
Court                              EL PASO           TX 79902James Mitchell             5005
North Bear Avenue     Apt. 92                    COPPELL           TX 75019William Clark
   4057 West Kennedy ExpresswaApt. 16                    CONSHOHOCKEN      PA 19428Jessica
Thompson          5788 East Tenth Court                                CROSS JUNCTION    VA
22625Anna Smith               8251 West Oak Boulevard                                ROME
     NY 13440Linda Smith               4528 North Wilson Court
WARREN            OH 44484Olivia Anderson            1011 North Taft Avenue     Apt. 67
        LOS ANGELES       CA 90006Madison Davis             8457 West Madison Road
                RYDAL             GA 30171Emily Robinson             7815 North Tyler Road
Apt. 107                  SAN JOSE          CA 95127Susan Mitchell             1639 East Hoover
Avenue                               ASHLAND           IL 62612Tyler Mitchell             1812
West Industry Street                          JAMAICA           NY 11435Samantha Stewart
   1834 North Boone Boulevard Apt. 99                    MENTOR            OH 44060Ashley Williams
          3754 West Pierce Street                                LUZERNE           MI 48636Ethan
Clark                4560 North Harding Boulevar                          VALDOSTA          GA
31602Grace Morison              5452 South Kennedy Court   Apt. 49                    MILAN
     IN 47031Ashley Thomas              5119 West Roosevelt Court
SURPRISE          AZ 85374Grace Robinson             4188 East Taylor Road
        CANTON            OH 44706James Clark               6623 South Clinton Street  Apt. 45
                INDIANA           PA 15701Kenneth Paterson          3577 North Ninth Street
Apt. 59                   TUPELO            MS 38801Thomas Young               6919 West Sixth
Street                               RANCHO CUCAMONGA  CA 91739David Moore               3562
South Calhoun Court   Apt. 13                    SCHAUMBURG        IL 60193Brianna Robinson
   718 South Maple Court      Apt. 37                    CINCINNATI        OH 45246Linda Thomas
          7224 North Taylor Street                               GLENDALE          CA 91201Dorothy
Thompson          4867 West Hoover Street                                PARKER            AZ
85344
```

Figure 3: A fully fixed file as viewed in a text editor.

```
Kenneth Mitchell          5547 East Eighth Court     Apt. 70                    KITTANNING        PA 16201
Elizabeth Thomson         7944 West Washington Court Apt. 44                    BATAVIA           IL 60510
Zachary Campbell          7982 East Taft Court                                  DINUBA            CA 93618
Steven Davis              3038 West Bush Avenue      Apt. 81                    DEXTER            OR 97431
Jacob Taylor              419 South Calhoun Street                              LANCASTER         WI 53813
Ryan Paterson             2888 East Carter Street    Apt. 2                     INDIANAPOLIS      IN 46260
David Paterson            651 South Kennedy Court                               EL PASO           TX 79902
James Mitchell            5005 North Bear Avenue     Apt. 92                    COPPELL           TX 75019
William Clark             4057 West Kennedy ExpresswaApt. 16                    CONSHOHOCKEN      PA 19428
Jessica Thompson          5788 East Tenth Court                                 CROSS JUNCTION    VA 22625
Anna Smith                8251 West Oak Boulevard                               ROME              NY 13440
```

Figure 4: A hybrid file as viewed in a text editor.

Fully fixed files contain no delimiters. This file type requires that the fields and consequently the records to be fixed in length (i.e., always a predefined number of characters). With no delimiters, each record is immediately adjacent to the preceding and following records and when viewed in text editor appears to have no structural formatting at all, see Figure 3.

A combination of these two, *hybrid files* are the third type of file considered. This representation contains characteristics common to both types of files (i.e., fixed length fields and records and a record delimiter), see Figure 4.

With respect to the layout inference problem, the proposed solution described herein is a statistical approach. This approach is well suited to this problem, a fact made evident when considering the data. As suggested, often the data is pulled directly out of an existing database, but how was the database populated? Coming from many sources, some prone to error and others generally unreliable, ensuring the correctness or validity of data is a problem in itself. Consequently it is important that layout inference allow for errors or incomplete data that may exist in the data; thus the statistical approach. Another important feature suggesting a statistical approach is the inability to perfectly model or describe all possible data values for recognition. For much data, the range of possible values is not closed (fully known beforehand) but rather is ever expanding. Consequently, it is better to attempt to identify most of the common values rather than including those that are new or otherwise exceptional. An example is personal names. Among all the possible valid values, there is a subset of widely used, common values that are most frequently used. Even though this subset exists, the complete set is an open domain, ever expanding due to the introduction of minor variations of existing values or entirely unique, often foreign, values. As the set is continually changing, it is difficult to completely model all the possible values. This concept is very powerful, providing reliable solutions in imperfect environments.

Within the overall statistical approach, other tools and methodologies are used to infer a file's layout. Of primary importance is the concept of an oracle. Oracles are

basically expert agents tuned for the recognition of respective properties within a file, and thus many of the computational steps associated with layout inference rely heavily on them. Once recognition is enabled via the oracles each of the individual properties associated with a layout are determined in an iterative manner. Flow-of-control within this process is relatively straightforward only branching based on the file type encountered. Because of the available delimiters in a fully delimited file, tokenization of the record structure is easily accomplished. All that remains is to identify the properties associated with the individual fields. On the other hand, the files with fixed length records, hybrid and fully fixed, field boundaries must be identified through some other means. In this case "tokenization" is accomplished by utilizing the oracles in a combinatoric search. Ultimately, the layout problem requires a variety of tools and methodologies all working together to obtain the various pieces of evidence necessary to best infer a file's layout.

At an introductory level, there are two defining assumptions associated with the layout inference problem: first is that the file is structurally consistent and the second is that the file contains a statistically insignificant number of errors and/or amount of noise. To be consistent, the properties at one position in a data file must reflect the corresponding properties at all other points in the file. So, for example, if a file uses one character encoding at the beginning and switches halfway through to another character encoding, the file would be inconsistent. Also, if a field is at the beginning of one group of records and at the end of another within the same file, then that file would be inconsistent. Inconsistency is considered separate from error or noise because it affects entire segments of data making it difficult to determine what is invalid and what is valid.

This uncertainty makes it challenging, if not impossible, to accurately represent the file as a whole thus rendering the output useless to later processing stages. On the other hand, because the results of layout inference are based on statistical evidence, an insignificant number of errors and/or amount of noise are allowed for within the data. For example, the presence of garbled or unrecognizable data in a few records out of many should not impair the layout inference results because the invalid records may be discarded or ignored in light of overwhelming evidence from the error free records. Error and noise are viewed as generally isolated events within a file's data. While it is difficult to draw an exact distinction between inconsistency and error, the two terms are used to indicate that problems at the file level, referred to here as inconsistency, are more difficult to address than problems at the record level, error and noise. These two assumptions are directly tied to the approach taken to the layout inference problem.

Using this definition of the layout inference problem, including the stated assumptions, an automated approach will be described throughout the remainder of the paper. This description provides solutions on how to identify the individual layout characteristics, detailing any further assumptions based off of the given assumptions, along with problems that may result and how to address them. In identifying the various components to the problem and their respective solutions an encompassing theory representing the layout inference process will be presented.

1.2 Thesis Statement

The thesis of this paper is that the process of determining the layout of an unknown file can be automated or semi-automated, provided that enough metadata is available about the content of such files. This thesis introduces the layout inference

problem and proposes a means of implementing an automated approach for converting an undefined file (i.e., a flat file, a file containing only sequential records of data, for which none of the associated properties are known) into a well-defined file (i.e., the same file except with defined attributes).

1.3 Organization of this Thesis

Using the problem description in this chapter, the remaining chapters provide details about the proposed solution: approach, implemented architecture, results, and conclusions. Chapter 2 describes existing work in related disciplines. Chapter 3 provides a general discussion of the approach taken to address the defined problem. Architectural details of the prototype that provide an implementation of the approach are given in Chapter 4 Implementation details are followed by Chapter 5 which contain experimental results and analysis obtained from the prototype. Finally, Chapter 6 provides conclusions and proposes potential areas of future work. Appendices are also provided at the end of the paper that provide an example of how the layout engine can be configured and file layout outputs generated by the engine.

2. RELATED WORK

2.1 Formal Languages

When working with languages, one of the first steps towards recognition is tokenization. Originally derived from computational linguistics, tokenization or lexical analysis is the process of separating data into meaningful units, tokens [3]. In the most common applications tokenization relies on special characters, typically some form of white space or punctuation, to delineate information. Thus, this important initial step can be considered to be relatively straightforward.

Layout inference also relies on tokenization in order to report the boundaries of data records and their comprising fields. With layout inference, the approach to tokenization depends on the file type. Fully delimited files are comparable to the types of data often encountered during tokenization because the available delimiters explicitly mark boundaries between the conceptual entities of records and fields. This then only requires the normal approach to define any meaningful units. For the other types of files pertinent to layout inference, hybrid and fully fixed files, a different approach must be taken. In this situation whitespace and punctuation often make up the contents of a field rather than separating them, consequently providing different information, see Figure 5.

Kenneth Mitchell	5547 East Eighth Court	Apt. 70	AUSTIN	TX	78727	5125551234
Elizabeth Thomson	7944 West Washington Court		AUSTIN	TX	78745	5125555943
Zachary Campbell	7982 East Taft Court		AUSTIN	TX	78754	5125554746
Steven Davis	3038 West Bush Avenue		AUSTIN	TX	78705	5125555153
Jacob Taylor	419 South Calhoun Street		AUSTIN	TX	78759	5125557821
Ryan Paterson	2888 East Carter Street		AUSTIN	TX	78730	5125559951
David Paterson	651 South Kennedy Court		AUSTIN	TX	78727	5125557516
James Mitchell	5005 North Bear Avenue		AUSTIN	TX	78722	5125558452
William Clark	4057 West Kennedy Expresswa	Apt. 16	AUSTIN	TX	78741	5125557661
Jessica Thompson	5788 East Tenth Court		AUSTIN	TX	78741	5125551529
Anna Smith	8251 West Oak Boulevard		AUSTIN	TX	78706	5125553238
Full Name	Address Line One	Address Line Two	City	ST	Zip	Phone

Figure 5: A hybrid file with identified fields.

This approach is further differentiated from typical forms of lexical analysis as identification is performed simultaneously. Commonly, tokenization is only concerned with identifying where data is (i.e., boundaries) leaving identification to subsequent processes, but in layout inference recognition of particular types within the data replaces the use of whitespace and punctuation as boundary indicators. Thus identification is a necessary side effect of tokenization. Lexical analysis is one concept relating formal languages to layout inference, an idea expanded to meet the unique requirements of the layout process.

Once a set of tokens is made available, a next step is to build an appropriate parse tree based on a predefined grammar [3]. This is relatable to layout inference where the layout (e.g., the field attributes, the record structure, and other properties that describe a file) can be considered a parse of a data file. The comparison is further evinced by comparing an ambiguous parse tree to the single record structure chosen from many candidates. In fact, the correct parse is chosen by statistical means not dissimilar to the process of choosing the correct parse of a probabilistic grammar [4-7]. Probabilistic grammars have relative weights corresponding to each associated production from which the correct parse is chosen. Among potentially several instances, determining the best parse is dependent upon the value and structure of the sentence being examined. If the fields represent the productions within a grammar, then the correct parse, record structure, is determined based on statistical evidence acquired during analysis. Considering probabilistic grammars can also lead to a discussion of grammar induction.

Layout inference bears further resemblance to Grammar Induction (GI) which is the problem of producing a grammar, regular or context free and often probabilistic in

nature, from a set of representative sentences [8-11]. Varying approaches have been developed, many that require a certain amount of human interaction [9] and others that attempt to provide a more automated solution [8, 10]. If the records of data are viewed as sentences and the layout as the probabilistic productions, then, at this level, the one problem can easily be viewed in the terms of the other. As corresponding details are considered though, differences begin to arise. First is the availability of the initial sentences. In GI, these are assumed to be ready to use whereas in layout inference, it can be expected that the records must first be identified and then separated. Also important is allowance for error. When an approach to GI includes error, negative examples, it is intentional and designated. Otherwise the samples are assumed to be a valid representation of an instance conforming to the produced grammar. Within layout inference, errors are not known beforehand and consequently must be handled as such. The inferred layout must still be correct even though the supporting data may be flawed. This is one of the strengths of the proposed statistical approach to layout inference. These examples suggest a relative commonality between layout inference and certain aspects of Formal Languages, yet also provide specific instances necessitating developing additional mechanisms for layout inference instead of these other technologies.

2.2 Information Extraction and Named Entity Recognition

Another discipline related to the layout inference problem is Information Extraction (IE); especially when it is considered via Named Entity Recognition (NER), an enabling sub-discipline. IE is concerned with distilling "structured data or knowledge from unstructured text by identifying references to named entities as well as stated relationships between such entities" [11]. For the purposes of this discussion,

"identifying references to named entities" is achieved through NER for which the "objective is to identify and categorize all members of certain categories of 'proper names' from a given corpus" [13]. Most references generally restrict "proper names" to name, address, and place content types but in certain instances have been expanded to include more specific examples such as biological names [14]. Interest and research in this field surged corresponding to the explosion of Internet-based information sources where most information is stored as free text or natural language. In order to make the data useful to available systems, pertinent data must be identified, extracted, possibly transformed, possibly cleaned, and loaded into a traditional storage medium such as a database. IE is concerned with the identification and extraction stages of data acquisition making it possible to pull data from sources previously difficult to process.

There are several technologies adapted to solve the NER problem, the most popular of which are: Maximum Entropy, Hidden Markov Models (HMM), and Decision Trees. Maximum Entropy accomplishes NER by first identifying several 'features' which are used to identify properties associated with the data [13]. Features exist as binary valued functions returning either TRUE or FALSE. There are many types of features including: lexical, dictionary, compound, and external system features. During training the features appropriate to a particular data category are automatically selected and weighted by the system. Given an input, the trained system first tokenizes the text and then runs the features against each token to see which features 'fire', or return TRUE. Given the resulting features and their corresponding weights for each token, a Vertibi search is performed to determine the most appropriate labeling, out of potentially many, for the tokens.

As Hidden Markov Models have been found to be useful in pattern recognition for speech recognition and NLP, they have been applied in varying ways to solve the problems associated with NER and consequently IE as well. Examples of such can be found in [15-17]. The methodology presented in [15] provides a solution to separating composite fields into the individual pieces. For example, once a data item has been identified as an address the system will further break it down into the house number, street name, city, state, etc… The Nymble system presented in [17] is similar to the maximum entropy approach presented above where the text is tokenized using white space, punctuation, and other available formatting elements after which each token is assigned a label. The difference is that a HMM is used to assign labels rather than feature selection. Finally the approach given in [16] builds on these techniques to enhance general HMM to incorporate dictionaries and allow for groupings of tokens. Whereas other approaches [13, 17] seek only to best label each individual token, the approach taken in [16] also considers groups of tokens, when appropriate, labeling them as a single unit.

Decision trees provide yet another alternative approach to the NER problem. The system described in [19] has defined three pieces of information necessary for labeling tokens: part-of-speech, character type information (i.e., token lexical properties), and special dictionaries (i.e., look-up tables or knowledge bases). Preceding both the training and testing stages these properties are determined for the corresponding data. Once acquired, they are used during the training stage to build the appropriate decision tree which reflects the associated properties of each applicable category type. In order to prevent deterministic labeling the leaves of the decision tree contain a probability

associated with each category type or label. This combined with Vertibi algorithm allows for the scenario where the most probable label for a token is not 'globally-consistent' and instead another less probable label must be chosen. Once the decision tree is acquired it is used in conjunction with the previously determined properties to label tokens during the testing phase.

In comparing the problem of IE with layout inference, there is a striking similarity: both are concerned with identifying important fields within textual data in order to define structured records. Beyond this step, the differences in the two problems begin to stand out. First, layout inference is not concerned with just field identification but is interested in other file properties as well (e.g., character encoding). Also, IE begins by assuming that the incoming data is ready to be tokenized using expected values such as white space and punctuation whereas layout inference does not. Even in fully delimited files (i.e., the most straightforward scenario with respect to tokenization) the appropriate delimiter must be identified before lexical analysis may be performed. In treating a delimiter as generic punctuation, some NER approaches could potentially be enhanced to handle fully delimited files, providing field identification; but this is only one property out of many associated with a file's layout. Another difference involves the text being examined. IE is concerned with unstructured, free text while layout inference is associated with files containing structured data records. This suggests that each will rely on separate characteristics in order to provide identification. IE relies heavily on textual clues associated with an available token and surrounding tokens (i.e., context) such as capitalization, punctuation, length properties, and even part of speech. Also, due to the nature of the source, the Internet, structural or formatting elements such as HTML

or XML tags along with generic labels (e.g., “NAME:” indicating a name field) are also allowed for. As layout inference works with records of data these features that are common within natural language are typically unavailable and are unreliable when available. Thus while there is a significant correlation between these two problems, their respective domains remain distinct.

2.3 Statistical Learning

In addressing the problem of NER, the described approaches were drawn from the field of Artificial Intelligence which indicates that the layout inference problem might contain certain commonalities with statistical learning algorithms. This is true, but only on a basic level. As indicated, the process to infer a data file’s layout relies on building statistical evidence which in turn suggests probable characteristics and structural organizations associated with the file. In the preceding paragraphs, this reasoning, among other similarities, relates this problem to the HMM and decision tree models used for NER and correspondingly probabilistic grammars. Also, relatable is the use of statistical evidence within learning algorithms. While the solution to layout inference presented in this paper does not make use of complete technologies available within the field of artificial and computation intelligence, it does rely on some of the underlying principles of statistical sampling. Not using artificial neural networks, genetic algorithms, or kernel machines was primarily based on the fact that these tools are too heavyweight (i.e., training and performance) for this problem without providing likely improvements. Given the provided solution, if desired, these types of technologies would be most appropriately applied in the task of field content type recognition, and thus the design architecture allows for their inclusion.

Besides the implementation of learning algorithms for recognition purposes, the statistical approach to layout inference also bears marked resemblance to several of the general properties found in statistical learning. Of note is the concept of the "hypothesis prior." To reduce over fitting, complex hypothesis are generally assigned a lower prior probability because there are typically more complex hypothesis than simple hypothesis [21]. Within layout inference, this can be seen directly in the ranking of field content types. In terms of layout inference, the field content types defined by limited domains, the simple hypothesis, are ranked higher than the field content types defined by very inclusive domains, complex hypothesis. Also, the structured file assumption inherent to layout inference could be seen to rely on the proof of the maximum-likelihood parameter learning concept which provides that the statistical properties of a sample is indicative of the corresponding properties of all the data [22]. Similarities between these two problem areas are generally limited to the properties of statistics, especially with regard to data sampling, and do not extend to the functional concepts of dynamic algorithm and parameter tuning provided by learning algorithms. The terms associated with layout inference, that were used for comparison (e.g., ranking and field content type) will be defined in the following section.

Beyond these basic, statistical properties, there are two other features commonly associated with artificial intelligence that are relevant to the layout inference approach described in this paper. First is the concept of search space pruning [23]. Particularly with respect to how the record structure is inferred, the organization of the processing stages is designed to filter out distracting or detrimental evidence in order to improve the inference process later on. Limiting the amount of evidence to be considered helps

restrict any unnecessary ambiguity. Organizing the system in stages effectively prunes the search space in that earlier decisions can rule out later false paths.

The second AI concept is the use of oracles. In the introductory remarks these oracles were referred to as expert agents because they are knowledgeable about particular domain elements. This comparison is drawn from the fact that each oracle is a distinct entity and can operate independently of or in conjunction with other oracles to produce a desired affect. Able to identify particular properties, each oracle can be considered to be an "expert" with respect to the associated attribute. Ultimately though, these agents are very limited in scope and may be defined as simple reflex agents which, ignoring history, produce output based only on the current percept [24]. Even the system they interact in, while multiagent, is only minimally so as the actions they produce have little impact on the actions of the other oracles.

2.4 Tabular Data Recognition

When records of data within a file are considered as two dimensional tables of information, another related discipline is that of table image recognition and OCR processing [25-28]. Of particular interest is the problem of column identification which is conceptually similar to the process of determining field position in the layout problem and the concept of automatic table understanding, where the labels of table columns are recovered, which is conceptually similar to identifying field types [29, 30]. Because the properties corresponding to a table are identified from an associated image in tabular data recognition, the supporting analysis methodologies are based on image recognition and consequently are not directly translatable to layout inference. In other words, this problem identifies records, columns, and fields by discovering the positional

correspondence of entries (e.g., all entries found at a single Y offset represent a record of data within a table, or all the entries found at a single X offset can be associated with a label entry found at the same X offset). Thus any similarity is essentially conceptual as identification is not inferred from the actual data values.

2.5 Text Mining

Text mining (TM) is also a related field, if only by association. Specifically, text mining may use the techniques and algorithms provided by IE and NER, among other areas, but the final goal is not the same [32]. Similar to data mining for structured data sources, text mining is focused on extracting previously unknown and unexpected trends or relationships from unstructured data sources [33, 34]. Also text mining is typically performed across a collection of texts rather than on a single document [16]. Another way to conceptualize the difference is that, in IE, the results identify a predefined set of content types, whereas in text mining, any information is to be determined from the data itself and is consequently not limited to finding fields of data. Thus while text mining might find layout inference, just as IE, useful in identifying and extracting particular pieces of information from data sources that is merely the beginning of identifying trends in the data.

3. LAYOUT INFERENCE APPROACH

3.1 Layout Characteristics

As mentioned in the introduction, there is a set of attributes associated with a data file that comprises a file layout. Before any other characteristics can be determined, the character encoding must be inferred. This provides the information necessary to interpret the data within the file, thus allowing further processing to be accomplished. After interpreting the data, any explicit structural elements, in the form of special characters, may be deduced. This also dictates how to proceed with layout inference, particularly which analysis steps are needed or appropriate for determining the record structure and which are not. Once the record length has been determined, the records may be logically separated and the field properties identified. As indicated by the suggested ordering, each property provides crucial evidence facilitating the inference of the remaining properties. Together, this information provides much information about a file and consequently is grouped together as the output results.

3.2 Initial Steps

Given the list of properties necessary to layout inference: character encoding, special characters, and record structure; the first two provide information about how to read the file, and thus must be identified first and in order. Once it is known how to interpret the contents of the file, further processing may be accomplished to determine the record structure. This is particularly significant because the existence or nonexistence of special characters indicates what functionality must be performed in order to be able to infer the record structure.

Due to the possible size of the files represented in this problem, exhaustive analysis is not desirable. Instead, a representative sample extracted from the beginning of the file provides the necessary characterization of the file. The assumption of consistency mentioned in the introduction indicates that the character encoding and special characters of this sample will remain constant throughout the remainder of the data file. The size of the sample must be large enough to ensure representative statistical results while being small enough to keep computational time within tolerable ranges.

Identifying the character encoding of a text corpus is nothing new. The process of Charset Detection has been accomplished before in many applications including web browsers [35]. The layout approach does not differ much from these approaches, providing a best attempt through the use of "statistics and heuristics" [36]. For the layout inference problem, the first step toward proper identification is to identify discriminating properties of each encoding. These are items that may be used to differentiate between two distinct encodings. Once accomplished, the contents of the file are sampled and the bytes tested for these properties. The results are then matched to the character encoding that they best represent. In this approach, the most important step is correctly identifying the properties associated with each encoding, as recognition subsequently becomes a straightforward process.

Special characters, such as commas or pipes, exist as additional structural information within the data. Acting as delimiters, these indicators may be contained inside a composite data item to separate the individual pieces of a field or may be located externally to indicate boundaries between fields of data. In either case, delimiters provide important information regarding how to properly separate the data and thus are

important when inferring a layout. Statistics and heuristics are used to determine what, if any, kinds of delimiters exist in the file. Given a predefined set of delimiters, where a delimiter is one or more characters in length, the sample is scanned once and counts indicating the number of occurrences in the sample of each set entry are tallied. Multiple non-intersecting sets, each representing distinct delimiter types (i.e., record, field, and text quote delimiters), must be allowed for as several delimiter types, each used to separate a unique structural element, may exist together in a single file. Given the counts associated with all the possible entries, the entry in each set that occurred most frequently is selected as the representative delimiter for that particular type. When the statistical count passes a corresponding, minimum threshold, the delimiter for the current type under consideration is saved and later reported in the results of the layout process. Otherwise it is discarded and no delimiter is reported for the current type. The unique threshold associated with each delimiter type is necessary in order to prevent errors or misidentification resulting from noise in the sample. Due to the dynamic nature of the program, the threshold is set as a small percentage of the sample size. This approach assumes that the layout engine, the program performing the steps necessary to infer a file's layout, knows the sets of delimiters before analysis begins.

3.3 Parsing a Record

The third component of a file layout is the record structure, which is minimally defined as the length of the record and the comprising field characteristics. While character encoding and special characters specify how to read the information contained within the file, the record structure designates where to look for specific units of

information. Before describing the analysis steps required to determine these properties within the context of the record structure, first consider the field characteristics and the tools used for recognition. Once available, these tools provide the means to identify each of the components of the record structure.

3.3.1 Field Characteristics

When characterizing a field, there are at least two important pieces of information: position and content type. A field's position may be represented as the start position and, depending on whether or not field delimiters are present, length of the field. Besides positional information, a field is also associated with a set of values that are associated with some content type. Content type is a category or label that provides identity to the corresponding data values. Assuming consistency, the field at a given position within a record must have a single content type. Each content type is associated with a domain: a set of valid values that the associated content type may assume. The rules or set of entries that define the domain are what distinguish one content type from another.

Just as the set of possible delimiters is specified before special character recognition, the field types to be recognized must also be designated and defined before being able to perform the remaining analysis required to parse the record structure. To this end the conceptual model of an oracle, whose sole purpose is to relate a domain to its respective content type, has been designed. In other words, oracles are structures that contain the domain definition corresponding to a particular content type, see Figure 6.

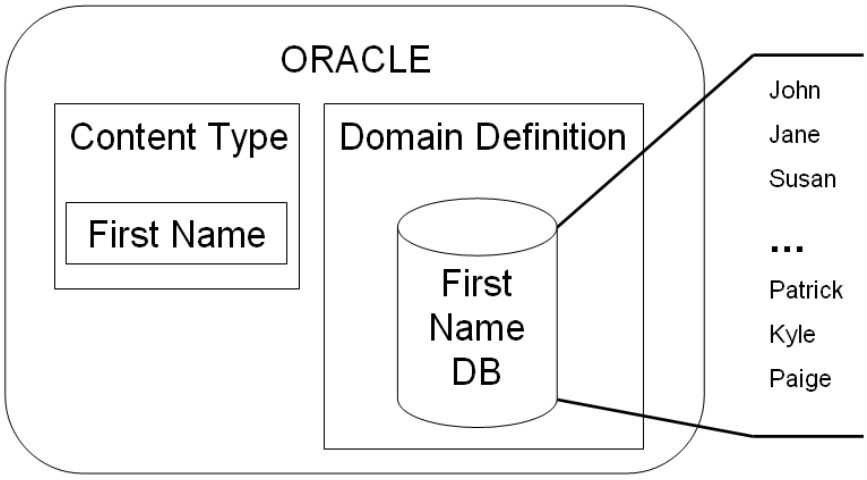

Figure 6: This oracle relates a list of first names to the "First Name" label.

When queried about a data item, an oracle determines if the item is contained within its defined domain and responds with an appropriate result. Thusly a data item may be related to a content type through the respective oracle's knowledge of a particular domain. It must be noted that the domains associated with human knowledge are often dynamic (e.g., continually expanding or the result of poorly defined, inaccurate or otherwise incomplete, syntax) and thus an oracle's corresponding domain definition may be incomplete, representing a best effort. This best effort is another argument for the use of statistical measurements in parsing the record structure. Together, a group of oracles provide the necessary functionality required to determine the remaining items of a file layout, first of which is record length.

3.3.2 Record Length

A record is a grouping of distinct yet associated pieces of information. These pieces of information are the comprising fields, and when grouped together they dictate the length of the record. The determined file type determines the best approach for

identifying the record length. Assuming consistency, when delimiters are present, the process of determining the record length is straightforward. Merely count the number of fields for variable length records and the number of characters for fixed length records that occur between record delimiters. When special character delimiters are not present, an appropriate substitute must be found to perform the role of delimiter completely changing the problem of record length identification. In this situation, the term "delimiter" is expanded to refer to anything that may be used to determine the boundaries between units of data rather than the previous definition of a set of special characters. As suggested in the preceding statement these delimiters are present in the data itself rather than external to it. Specifically the positioning of certain fields provides the delimiting information.

To use a field as a delimiter there are three steps that must be accomplished. First a field of an appropriate content type must be chosen. This is the most important step primarily because when considering the many content types that may be present within a record of data, some provide much more reliable results than others. First the content type must exist in the examined data. It would be pointless to use a content type such as personal first name for records about car parts. Once the viable content types and their associated fields have been identified, the next step is to determine which of these will occur in a statistically significant number of the records. It is important to avoid fields that may contain blank values, and that are often so, as a disproportionate number of blank values may skew the required statistical evidence. Once it is known that a particular field will exist in the data and will not frequently contain blank values, another important criterion to consider is whether or not the field is easily identifiable. There are

at least two properties associated with a field being easily identifiable: the field's content type is not readily confused with other content types and it is not too complex as to make finding the field boundaries, especially the start position, difficult. The first property is a result of domain overlap. The domain overlap problem is a significant sub problem associated with the layout inference problem. It is caused by multiple domains that share common values. Given two domains *A* and *B*, these domains are said to share a common value if there exists an entry in *A*, *a*, such that *a* is equal to or is a substring of an entry in *B*, *b*; or vice versa. These two possibilities are based on the file type. Since field boundaries are known beforehand in fully delimited files "common values" are based on equality. When considering fixed files, since the field boundaries are not given, "common values' are defined by equality and substrings. A good example for this particular discussion is zip codes. While relatively easy to find, because it is a number, a zip code is readily confused with other number fields (e.g., street, phone, and account numbers). Thus while phone numbers are ten digits in length and zip codes only five, because zip codes are often a substring of valid phone numbers these two domains share many common values. Consequently, with respect to finding record length, zip code is not a desirable field to base this analysis upon. The impact of this problem on the various analysis steps performed during layout inference depends on the sample from the input file and also the degree to which distinct content types overlap. Finding field boundaries is a problem common to composite fields, see Figure 7. A composite field is one whose type is a combination of several simple content types. This is a feature characteristic of database schemas where several individual yet related components are grouped together (e.g., full name and address).

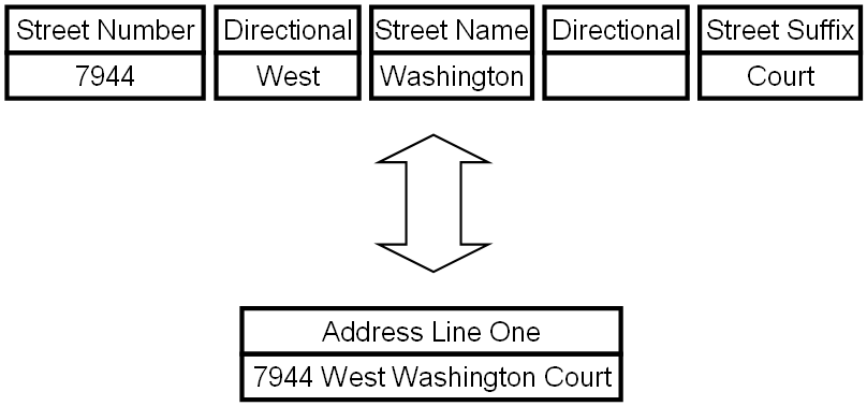

Figure 7: A composite field is multiple simple fields grouped together.

Composite fields are potentially unhelpful because the accurate positional information required for using the data as a delimiter is not always obvious. Together, these criteria ensure adequate statistical results from which to derive the record length.

Once a content type has been chosen, the final two steps are computational in nature. First is to identify where fields of the associated content type begin. To this end, a combinatoric approach has been developed which enumerates all possible start positions and lengths. This approach will be further explained in the following sections as the means by which field type and position are elucidated. The oracle corresponding to the chosen content type is used within this combinatoric scan to determine where the associated fields occur. As mentioned, the oracles provide the means to test whether a sequence of characters from the data is valid within the confines of a particular content type's domain. When an oracle returns TRUE for a particular sequence of characters, the start position of the sequence is recorded. Together, these start positions provide enough information from which to determine the record length. There are two types of errors that often occur with this approach: false positives and missed fields. The first is when another field is mistakenly identified as having the same content type and the start

position is recorded, and the second is when a field should be identified but is not. Both are a result of the inexact nature of domain definitions and all that entails. To allow for these errors, and potentially others, it is appropriate to make a statistical determination rather than an exact one.

The final step is to identify the record length. Two methods were considered as a solution to this problem, both of which rely on the property that the records are fixed in length. The first was to find a common offset between field positions. The idea behind this approach is that the offset, once found, would be the record length as it represents the distance between fields of common type. This idea presents several difficulties in light of the characteristics representative of the data file layout inference problem. First, false positives and missed fields can significantly impair the results. These errors can provide many varying offsets making it difficult to choose the correct length, impairing the statistical quality of the result. Though it is possible that a statistical approach can overcome these difficulties the problem is magnified when fields are duplicated. It is quite feasible that a single piece of information might appear multiple times in single record. While not the best normal form, it is not uncommon. When this occurs, multiple offsets will be reported with the same statistical evidence causing even more uncertainty with respect to the correct record length. This problem is also solvable, but a more straightforward solution exists.

The other approach that was considered, the one implemented in the associated prototype, finds the record length by examining multiple possible lengths until the field positions logically line up, Figure 8. This approach starts with some initial record length, possibly a length of one, and incrementally considers record lengths until the positioning

of the fields found in the previous step organize themselves with respect to the hypothesized record length (i.e., they logically line up). When this organization is deemed to exist, the procedure declares victory and returns the current guess as the determined record length. A threshold value is used to indicate when the fields line up. This value should be set high enough to avoid the possibility of random organizations caused by false positives and low enough to allow for a certain number of missed and blank fields. This statistical approach is not hampered by errors or duplicates as the other approach is. In fact the positional results associated with these problems provide little, if any, interference with the evidence provided by the actual fields. Both approaches make assumptions based on the content type selection criteria mentioned in the second paragraph of this section.

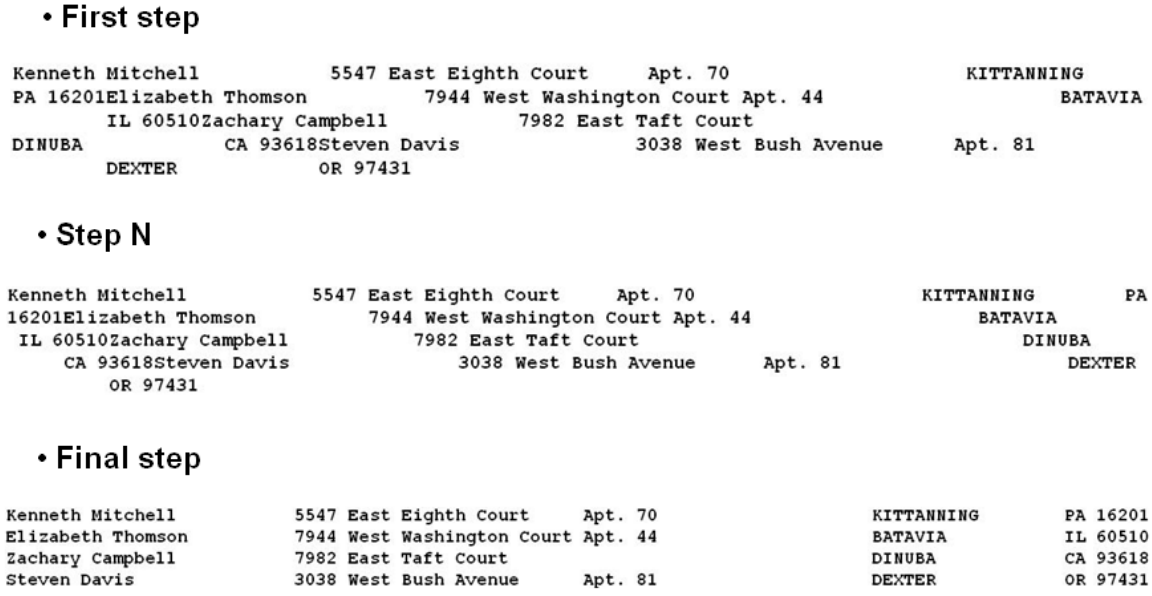

Figure 8: Example of fully fixed record length identification using full names.

3.3.3 Field Content Type

Incrementally more and more information has been ascertained about a data file through the previously described analysis steps. Each preceding property either makes it easier to or is necessary for determining the current element of a file layout. The last stage it is no different. By first determining the character encoding, it is known how to interpret the data in the file. Then by identifying special characters, specifically delimiters, it is possible to recognize any external structural clues provided by the data source. These characters along with the determined record length indicate how best to divide the data into logical components (i.e., records and possibly even fields). Together, all this information makes it simpler to parse the record into its comprising fields.

Parsing delimited files requires the least amount of computation of the three file representations considered. Since field delimiters indicate field boundaries, and thus position, the only remaining step is to determine type. Using the available delimiters, the data may first be broken up into records and then separated into fields. The end result of this division is multiple sets of values each corresponding to a unique position within the record. The next step is to assign a content type to each set of values. To accomplish this, each set of values is considered individually. For a particular set, each available oracle is considered in turn and a total count acquired that indicates the number of valid entry values according to the oracle's respective domain definition. In other words, given a set of values, $V = \{v_1, v_2, \ldots, v_{n-1}, v_n\}$, and an oracle, o, the total count is the result of $\sum_{i=1}^{|V|} o(v_i)$; $o(v_i) = \{1$ if v_i is considered valid by oracle o, 0 if not$\}$. Once a count is achieved for each oracle the process is repeated for each set of values. These counts, which correlate a field description to a set of values, provide the statistical evidence used

to infer the correct field designation among potentially many. A single field description, along with its respective statistical count will thus be referred to as a potential field (PF). From many of the papers describing the areas of IE and NER, it seems plausible that some of the presented solutions could solve or be adapted to solve this particular portion of the layout inference problem: identification of field content type in delimited files. Essentially, the available delimiters provide enough information to easily tokenize the data, but problems might arise from reliance on contextual clues of surrounding fields common to free text documents.

Parsing a fully fixed or hybrid file requires a different tokenization approach to that just described. Particularly is the importance of determining field position, as there are no special characters indicating field boundaries. Fortunately this can be accomplished simultaneously by comparing the oracles against the data in a combinatoric way. Since the record length is now known, the data can be separated into records. It is helpful at this stage to at least know the record boundaries, because the statistical results derived from the combinatorial scan of the data are interpreted within the context of the record superstructure. On a side note to be considered later, this scenario also reduces the worst case run time because now the time to perform a brute force search of the data is primarily dependent upon the length of the records rather than the length of the data sample, which can be significantly longer. Once a list of records is available the combinatoric or brute force analysis begins.

Each field position and content type is identified by testing all possible start position and length combinations. Slightly tweaking the idea used for delimited files where a field represents a set of values offset by delimiters within a list of records, in this

case a field represents the set of values within a list of records at a given position as specified by a start position and length pair. Thus for each start position and length pair, the sequence of characters at the specified position within each record represent the entries in the set of values corresponding to a field. In the same way as with delimited files, the entries in the set are examined by an oracle and the number of valid entries recorded for subsequent analysis stages. This is repeated for each available oracle. Once completed, this process is repeated for another position until all possible positions have been enumerated. The following figure represents one iteration of combinatoric analysis. In this example the "First Name" content type is considered and for a single length each start position is examined. Once completed, the length will be altered and the process repeated until all lengths have been considered for this particular content type.

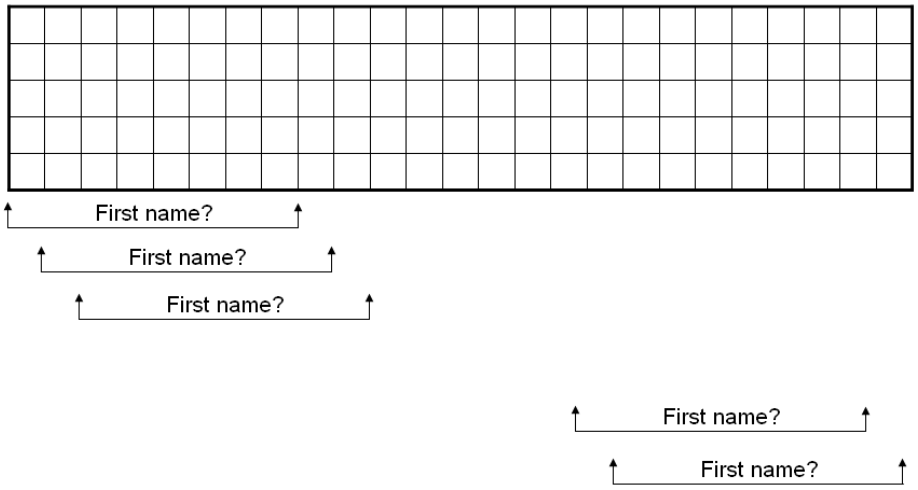

Figure 9: Depiction of combinatoric analysis.

Once the necessary counts have been tallied, the correct position for each field must be determined. This requires a correct understanding of the results produced by the combinatoric analysis, showing why this approach is appropriate for fixed and hybrid files. When data is stored as fixed length fields a length is chosen for each field

appropriate to the values that will exist in the respective field, a length that is long enough to contain all or almost all foreseeable values. When an actual value is shorter than the maximum length prescribed for the field, the remaining characters are unused and must be filled, typically with white space. For values that exceed the prescribed length the value must be truncated. On a side note, the latter case is another reason why error must be allowed for in the layout problem. This characterization of the data suggests a general trend that may be taken advantage of. In graphical terms, as longer and longer lengths are considered from a single, field start position, the number of valid values defined by the current position will increase until reaching some apex after which the number will begin to drop, Figure 10. While increase and decrease in the number of valid entries may change gradually or quickly, and while the apex may be a single point or a plateau, the general trend describes the representative structure of the data and can be used to infer the positional elements of a field. Specifically, by counting the number of valid entries for each start position and length pair, the results of the combinatoric analysis provide a graph for each start position that correlates lengths to valid counts. For each content type, possible start positions are chosen by finding which have associated valid counts large enough to indicate a field. The length appropriate to a given start position is indicated by identifying the last length before the valid count begins to drop significantly. Using this information the proper position can be determined and a representative potential field (PF) created.

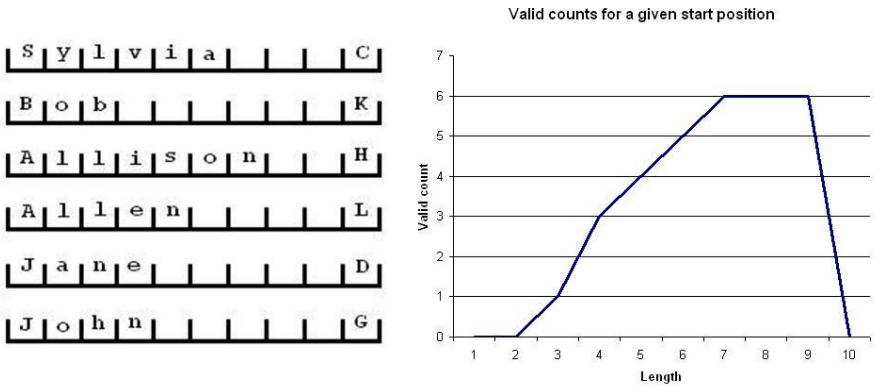

Figure 10: Example of an apex with a plateau.

Composite fields must be handled in a slightly different manner. Often in database schema definitions, highly related simple fields are grouped together into a single composite field. This is a common occurrence with content types such as names and addresses. Considering the valid counts for a composite field, it is possible for multiple apexes, plateaus, to occur for varying lengths at a given start position. Because an apex is the maximum valid count corresponding to the length from a particular start position, to be in this group of apexes each individual apex must have the same valid count; otherwise it is not an apex. As longer lengths are considered, a value from a composite field may alternate between valid and invalid several times depending on the organization of the simple fields within a composite field. This causes valleys between the several apexes. Due to this fact, the appropriate length of a composite field is the length corresponding to the rightmost apex, the longest length with the maximum valid count. Using the rightmost apex ensures that the entire composite field is included in the positional element of the field information. This is merely a generalization of the single apex encountered with simple fields.

3.3.3.1 Verifying the Results

The combinatoric analysis just described scans the record data in a horizontal fashion. The evidence acquired by examining each row individually can be incomplete as it is unable to account for trends that may only become apparent when the field values are viewed all at once. This implies the need for a vertical, field-based, analysis. Using the PF as indicators, it is possible at this point to look at the complete set of values for an individual field. While possibly not necessary for all content types, this form of analysis is often valuable in addressing part of the domain overlap problem. It is especially applicable in recognizing when the set of values corresponding to a PF only contains a small number of unique entries all or most of which are shared by multiple domains. An obvious scenario is when a file is sorted on a field and thus all of the entries in the set are equal. In many cases random sampling can alleviate this problem, but for small files it is possible that each record contain the same value for the field being sorted on and thus this eventuality must be allowed for. When all of the entries are equal in value, a corresponding flag can be set thus communicating a fact that may be useful later on. In the case of more than one unique entry value in a set, it is possible to ascertain the correct content type by determining for which domain the entry values better represent a normal distribution of data. For example, if the set of values for a field along with occurrence count was {a - 15, b - 3, c - 2, d - 1} which returned a PF for both content type *I* defined by domain {a} and content type *II* defined by domain {a, b, c} then content type *II* is a better fit. This is because even though the value {a} occurs most frequently, enough to potentially identify the field as content type *I*, the values of {b, c} indicate that in fact content type *II* is most appropriate. While content type *II* also has a higher valid count,

this contrived example also attempts to show the importance of vertical analysis. When domain overlap causes multiple PF to conflict, vertical analysis can be used to indicate if any of the PF are not plausible or at least less so.

A previous figure, Figure 5, helps to show some of the issues for which vertical analysis provides additional information. First, consider the "City" field which contains a single value, "Austin," for each record. This would go unrecognized by the horizontal, combinatorial, analysis and the field could equally be a first name, street name, or city. By performing vertical analysis the fact that only a single entry exists could be marked and handled later on as a special case. Another circumstance would be the "2" in the "512" area code of the phone number field. Individually these could be considered to be name suffixes associated with a person who has the same name as their parent. By scanning the column no statistical distribution is encountered and the column is determined not to be a name suffix. These are just two examples of the use of vertical analysis.

3.3.3.2 Consolidating the Results

Upon completing the preceding analysis steps several PF have been identified. As mentioned, these represent hypotheses identifying the position and type of fields within a record. Given the combinatoric approach, there are two problems to resolve at this stage. The first is conflict resolution. Because PF of varying content types may have been assigned to the same position within the record, a decision must be made regarding which is best suited. The second is the identification and elimination of false positives from the results. As the name implies a false positive is a PF that has been incorrectly

assigned to the record structure. Once these two issues have been addressed a final representation of the record structure can be made.

3.3.3.3 Conflict Resolution

Because of the problem of domain overlap between content types it is possible that the positions of multiple PF, each of a different content type, will completely or partially conflict with each other. Conflict is when the positions of two PF, X and Y, overlap each other such that start positionX ≤ start positionY ≤ (start positionX + lengthX) or vice versa. For content types with which domain overlap is a significant problem, it is helpful to perform one final step to gather more evidence before choosing among conflicting PF. This step is to intentionally restrict the domain definitions present in the respective oracles as a means by which to reduce the overlap between the domains – Figure 11. Once the domains have been restricted, the sets of field values corresponding to the positions of the appropriate PF are scanned. This scan will produce a secondary count to associate with each respective PF providing additional statistical evidence to better differentiate among the conflicting entries.

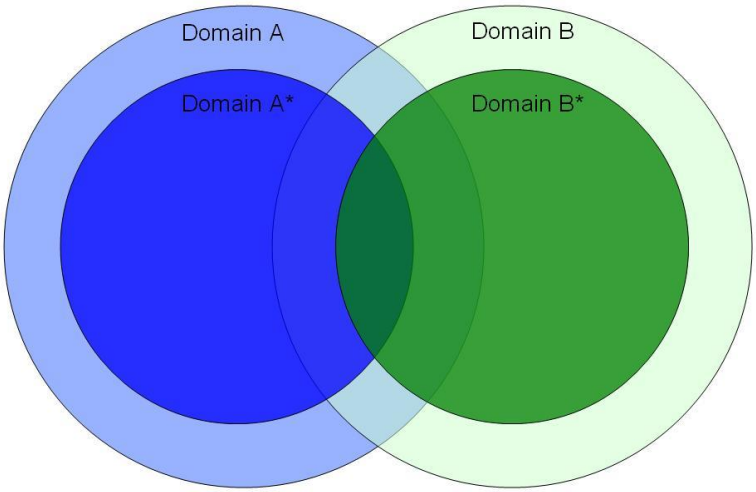

Figure 11: Figurative depiction of domain reduction. * indicates a reduced domain.

To determine the fields in the record structure, a relatively simplistic approach has been adopted which provides satisfactory results. The set of PF for each content type are each considered in turn. Upon consideration, each PF in the current set is either selected as the best candidate for the corresponding position in the record or discarded. To be selected as the best candidate, the current PF must either not conflict with another PF already chosen as the best candidate or must prove to be a better candidate than the PF it conflicts with. When a conflict occurs, the evidence previously gathered is considered along with the empirically determined reliability of the associated content type's domain definition thus indicating the best candidate between two PF. The statistical evidence exists as the primary count determined during the combinatoric stage and, when available, the secondary count acquired after reducing the designated oracle's domains. When a PF, X, is determined to be a better candidate than the existing best candidate, Y, X will replace Y as the best candidate and Y will be discarded. After considering each content type, the final PF considered best candidates are assigned to the record structure as fields. Because the records in the data file may contain content types unrecognizable

to the layout engine, any ranges of characters not corresponding to a field definition are marked as unknown. Given the available evidence, this approach provides a result better than or equal to all other possible results. Assume that there exists another possible organization that is strictly better than the one chosen. This assumption indicates that a discarded PF is a better choice than the final candidate it conflicts with. This is a contradiction though, because in order for a PF to be a final candidate it must be a better candidate than any PF it conflicts with. Better, more correct, solutions may exist, but they are not indicated by the gathered evidence.

3.3.3.4 Result Enhancement

Once conflict resolution is complete, the set of resulting candidates have been assigned to represent the fields of the record structure. Given this set of fields, one final step remains: to enhance the results by removing false positives. This analysis stage has been deferred until this point in order to prevent unnecessary work. During conflict resolution many false positives were discarded in the process of choosing the best candidate among conflicting entries. Also, in restricting the set of fields the analysis performed at this stage becomes more reliable, because close approximation to the actual contents of the record improves analysis. This approach includes cross-reference and contextual analyses.

Conceptually, cross-reference analysis is a means by which to identify which fields reflect a defined relationship among domains of different content types and which do not. Thus, cross-reference analysis is accomplished by referencing the values of two or more fields within a single record and testing whether or not they conform to a defined relationship. In other words, cross-referencing of field values is possible if there exists a

mapping from the domain of one content type to the domain of one or more other content types. This form of analysis is often used during database integration to enhance the results of field content type identification [38]. Any group of fields determined to be related by a particular mapping can be confidently included in the final record structure. The remaining fields, those not assigned to a valid grouping, are potentially false positives. Identifying false positives in this manner assumes that the values of any fields corresponding to a content type associated with a defined relationship must map to the values of a field corresponding to another content type in the relationship. This assumption implies that given two relatable fields, *A* and *B*, for each value in *A* there must be a value within *B* that the value from domain *A* maps to and vice versa. If this is not true, then the extraneous fields are assumed to be false positives. Cross-reference analysis attempts to find relationships among the values of certain fields. Conceptually similar to vertical analysis, cross-referencing recognizes trends in the values of fields in order to determine which PF are best suited for the result.

As an example, candidates for cross reference analysis are the content types: zip code, city, and state. Because the zip code domain contains only numbers, the associated oracle can be prone to report false positives. In performing cross reference analysis, every possible city, state, and zip code combination is examined. From this, a set of correlated fields is returned indicated by extremely high correspondence among the respective field values. Only the fields that are reported are considered to be valid whereas those that remain are determined to be false positives. In order to provide a clearer example, let three fields be identified as zip codes: field *1* is the correct zip code field, field *2* is part of a phone number, and field *3* is part of an account number. For

simplicity let there be a single city and state field. The possible combinations of fields are: (city, state, field *1)*, (city, state, field *2*), and (city, state, field *3*). When cross-referenced the first combination should return a very high correspondence as field *1* is the actual zip code field associated with the city and state fields. The other two combinations should return an insignificant, if not negligible, correspondence. Subsequently the first combination is returned as valid and the zip code fields determined to be PF, fields *2* and *3*, are removed from consideration.

Another form of analysis pertinent to this stage is contextual analysis which is common, and typically required, in most IE and NER systems. Within the context of data file layout inference, contextual analysis refers to the relative nearness of fields of similar content type and as such takes advantage of any organizational features inherent to the record structure. Context often does not exist when considering the data representative of the layout inference problem. This is due to the fact that fields are often added to databases, the typical source of layout inference data files, in a haphazard way usually being appended to the end of a record. For example if a database was expanded to include middle names the field would often be added to the end of the supporting schema completely isolated from the other, existing name parts. This could cause two fields of similar content type to be relatively distant from each other within the record. On the other hand, when available, context can provide useful evidence indicating false positives. As this type of analysis is often not available, it must be used with caution because relying on context will often prove to be unreliable for layout inference.

3.3.4 Results from Evidence

Just as in the related fields of IE and NER along with parts of learning algorithms and database integration, layout inference relies on several kinds of statistical evidence in order to produce results. Considering layout inference, this evidence begins by counting how many records contain values corresponding to a defined type at a given position. These potential fields are just the beginning. From there vertical analysis recognizes trends, specifically values representing small domain intersections, unattainable from a horizontal scan potentially reducing the set of PF to be decided among. Conflict resolution further reduces the set of PF to the subset that most accurately, based on the available evidence, represents the record structure. From this subset further enhancement is performed by attempting to identify and remove false positives. Each step represents the acquisition of different types of evidence in order to systematically isolate and determine which potential fields are most appropriate as part of the record structure.

3.4 Reporting the Results

Upon obtaining the desired results, the remaining step required for layout inference is to report the information. Due to the nature of layout inference, the results include both the final layout specification along with much of the information derived in producing the results. This reflects the understanding that the statistical results of layout inference may be imperfect, particularly when the assumptions about the data are not met. Thus the results not only indicate the properties determined to be most appropriate based on the data, but also all possible candidates from which the result was chosen. Along with the gathered evidence, many of the assumptions can also be specified by

indicating what properties were predefined (e.g., delimiters). Including this information allows a user to choose how to interpret the results, especially if they are found to be inadequate. Ultimately, the physical report should be organized such that all of the necessary information can be presented in an easily recognizable manner as determined by the end user.

4. PROTOTYPE DESCRIPTION

4.1 Introduction

With the given approach as a guide, a layout inference engine prototype was developed using the Java programming language. The prototype explores the issues discussed in the previous section and thus serves as a proof of concept. The prototype is sequential: starting at a single entry point and iterating through several analysis steps after which the output is produced and the program returns. The remaining segments in this section explain the various components and design details of the layout engine which follow the flow of control shown in Figure 12. The basic flow of control used to determine the record structure is expanded in Figure 13, indicating the several sub problems to be solved in order to identify the record structure. While the overall approach is the same, the implementation details for each file type are unique. A feature further discussed throughout this chapter, this is indicated in Figure 12 by the branch on file type.

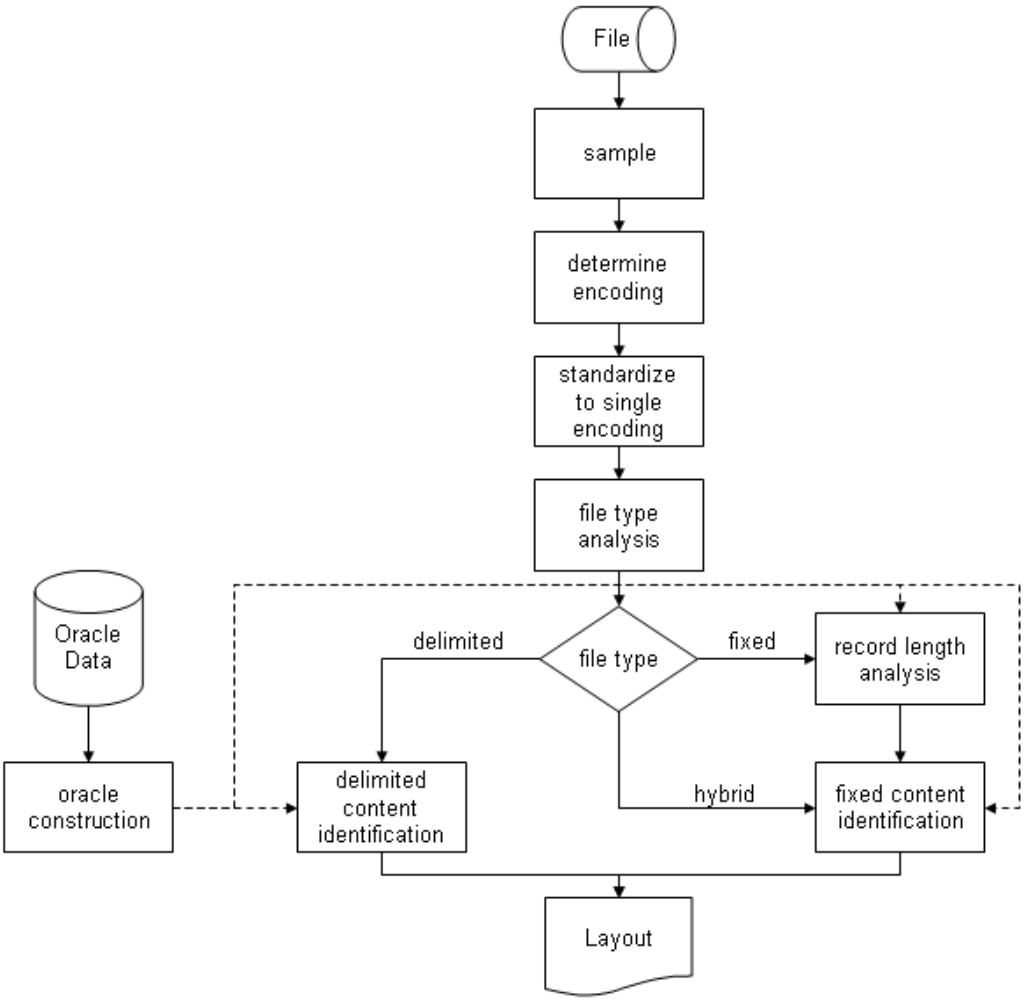

Figure 12: Layout engine flow of control

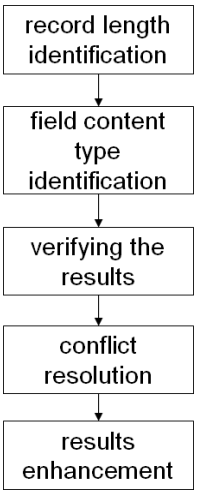

Figure 13: Record structure identification.

4.2 Prototype Invocation

For ease of use, there are two ways to invoke the layout engine. The first is from the command line and the second is as a web service. There are two parameters to the program, the location of a configuration file and the data file whose layout is to be determined. The web service functionality was included as a convenience and makes the assumption that the data file can be accessed by the server. This assumption will be explained later. The single purpose of the entry points is to begin analysis and return the compiled results.

4.1 Setting Parameters

Upon invocation, the first step is to setup the layout engine. Dispersed throughout the program, there are many options, thresholds and heuristics that may be specified. Beyond these general parameters and in order to facilitate extensibility, it is also possible to define what oracles are run and the associated parameters of each. Together the parameters specified in the configuration file allow a user to tune the layout engine as necessary. Written in basic XML, an example of the configuration file is given in Appendix A. This example provides default values for each of the following described parameters.

Encountered first in the configuration file are "global_parameters". The options in this group are dispersed throughout the program being used in the various analysis stages.

- "PrintTextOutput" – a Boolean flag indicating the type of output the prototype should return. If set to TRUE then the prototype will return the

output in a more human readable format otherwise the output will be in XML which is much more verbose.

- "EbcdicPercentage" – this percentage establishes how many bytes from the sample must have the first bit set in order to accurately decide between ASCII and EBCDIC character encodings.
- "RecordsToTest" and "AcceptableRecordCount" – These are heuristics limiting the number of records considered during each iteration of the combinatoric stage. Given a start position and length if there are not at least "AcceptableRecordCount" valid entries out of a distributed sampling of size "RecordsToTest" then the current start position and length are ignored and the next pair considered (i.e., a sampling of the sampling). The reasoning for this is based off the idea that when looking at data a human does not initially consider all the entries, but rather just a few. This small set determines whether or not the remainder should also be considered.
- "OptimalRecordCount" and "SamplePartitionCount" – These parameters determine the sample size extracted from the data file from which the record structure is determined. Once the record length has been determined the file is re-sampled and "SamplePartitionCount" blocks are read from the file which together will approximately contain "OptimalRecordCount" records. This re-sampling is the reason the data must be available to the web-service server. Only an approximation can be made for variable length records and thus the total number of records may be slightly different than "OptimalRecordCount."

- The seventh parameter, "HeaderRecordsToSkip," provides for the fact that a header record may exist in the data file and that when sampling to determine the record structure this special record should be ignored.
- "SampleSize" – specifies the size of original sample from which the character encoding, special characters, and record length are determined. It is helpful that the sample be acquired from the beginning of the file in case the file is fully fixed. If obtained from a random position within the data file the positioning of fields within the sample would likely be skewed making it difficult to determine the appropriate record length.
- "LineUpPercentage" – specifies how many valid entries must occur at a given start position for the fields to line-up and is used to indicate when the correct record length has been found.
- The final three options "RecordDelimiter," "FieldDelimiter," and "TextDelimiter" provide the possible sets of values corresponding to the respective delimiter types that the prototype should look for. The characters are given by their ASCII representation (e.g., the character '|' is indexed in ASCII by the decimal value 124). Each individual delimiter in a set is separated by ';' and the possibly multiple characters in a delimiter are separated by ','.

The remainder of the configuration file is used to define what oracles will be run by the prototype. First within each oracle configuration definition is the qualified name of the oracle from which the prototype dynamically loads the oracle class at run-time. Also included is the second parameter, "Rank," which is an integer value specifying the

reliability of the associated oracle. This number is relative, where the larger the number the more reliable the content type of the corresponding oracle is determined to be. These first two options are used by the container class to first load and then to order the oracles.

Any remaining parameters are grouped together and define the oracles themselves. While there are many oracle specific parameters, there are four parameters shared by all oracles:

- "TypeName" – the label associated with the content type used for identification.
- "MaximumLength" – a heuristic used to reduce the computational requirements of the combinatoric stage. Empirically determined, each content type has some upper bound with respect to its length which is less than or equal to the record length. By stopping at this defined length it is possible to reduce the number of start position and length pairs considered during combinatoric analysis.
- "MinimumThreshold" – a boundary percentage indicating which PF should be passed on to the conflict resolution stage and which should be discarded. If the valid count associated with a PF compared to the total number of records is less than this percentage then the PF is discarded.
- "Grouping" – a bitmask used to indicate logical groupings among oracles (e.g., street number, directional, street name, and street suffix could all be considered part of one grouping, a grouping of address line components). This grouping information, further detailed in Section 4.5.3, is used to

determine what statistical evidence is most appropriate to resolve conflicts among PF.

4.2 Character Encoding

After the engine's parameters have been set the prototype begins to follow the flow of control provided at the introduction to this section. The prototype has the ability to identify the encoding from three possible candidates: ASCII, EBCDIC, and Unicode in its three variations UTF-8, 16, and 32 [39]. As mentioned in section 3.2, the first step is to describe the unique characteristics of each encoding. Once these are available, the data sample is scanned testing which of the characteristics is most prevalent and thus which encoding is most appropriate. The ordering of the tests can be important as some of the discriminating properties are not entirely distinct. For example, it is straightforward to determine whether the sample is ASCII or EBCDIC, but choosing between EBCDIC and Unicode is not so obvious. In the latter case, Unicode is tested for first, but if those tests fail then EBCDIC becomes a more plausible candidate.

First, UTF-32 and 16 are considered. For most valid characters, characters within some language's alphabet, these two encodings will contain a null byte in their two or four byte encodings. This is particularly true of UTF-32. So if null bytes are consistently encountered at the beginning, or end depending on endianness, of a group of bytes then the sample is probably one of these encodings. After testing for null bytes, ASCII is considered next. ASCII encoding is distinct from the other two in that the most significant bit is never set. Therefore, if a very high percentage of the sampled bytes do not use this bit, then ASCII is chosen as the encoding. If the sample is still unrecognized, then UTF-8 is considered. UTF-8, as a variable length encoding, has a format definition

that the encoded bytes must conform to [35]. Thus, upon recognizing this format, possibly along with an optional, prepended byte order mark, the prototype chooses UTF-8 as the character encoding. If the character encoding is still undetermined, EBCDIC is considered next. This test assumes the encoding to be EBCDIC and converts it to ASCII counting the characters that map correctly. If a very high percentage mapped correctly, then in the absence of indicators for the other encodings, the sample is determined to be EBCDIC. If not EBCDIC, then the sample is assumed to be UTF-16. While it is possible that UTF-16 will contain null bytes, it can not be assumed, particularly for foreign character sets. Without any other distinguishing characteristics, UTF-16 is left as the default.

4.3 Delimiters and File Type

Once it is known how to read the data in the file, the next step is to determine if any delimiters exist in the sample. This step is important because delimiters provide clues about the structural representation of the records of data within the file and thus indicate how best to examine the records. There are three types of delimiters considered: record, field and text. Text delimiters are merely recognized and reported while the record and field delimiters determine the type of file being considered and thus how the analysis of the record structure will proceed. The first type of file is a fully delimited file which contains both record and field delimiters. In this type of file the fields and thus the records may be variable in length because of the special characters explicitly indicating the respective boundaries. On the opposite end of the spectrum are fully fixed files. In files of this type, there are no delimiters of any kind and field and record boundaries are based on fixed lengths. Thus, when one field or record ends, another begins immediately

afterwards. The final type considered is the hybrid file which is a combination of the other two types. This file contains fixed length fields with no field delimiters but does contain record delimiters, such as a line feed, at the end of each record.

The representative sets of delimiters are specified by the user in the configuration file just described. Once available, these sets are considered in turn starting with the record delimiters. The sample is scanned once and a count corresponding to each entry is incremented whenever the respective entry is encountered in the sample. The prototype allows a delimiter to be comprised of a sequence of characters, such as a carriage return followed by a line feed, or an individual character. Once the counts are tallied, they are compared to a configurable minimum threshold to allow for noise and those that are greater than this empirically determined threshold are compared against each other. The delimiter with maximum count is selected and is recorded as the corresponding type of delimiter. Of note is the special case of string literals typically indicated by the characters " or '. These are used to enclose delimiting characters that are actually part of the data, such as a comma separating the first and last names in a full name field. When a string literal character is encountered any following data is passed over until a matching character is found. This logic assumes that for each opening string literal character there is a matching closing character. Using this approach the delimiting characters can be quickly identified and used to determine how best to proceed with respect to analyzing the file.

4.5 Oracles

Before any further analysis can be performed, the oracles must be defined. They were previously described as a black box used for recognition of data corresponding to

associated content types, but now the actual classes and corresponding interfaces will be described in order to clarify their programmatic design. Of importance in such a discussion are three constructs depicted in Figure 14: the oracles, the container class that acts as a wrapper for all of the oracles, and the interfaces that the oracles implement.

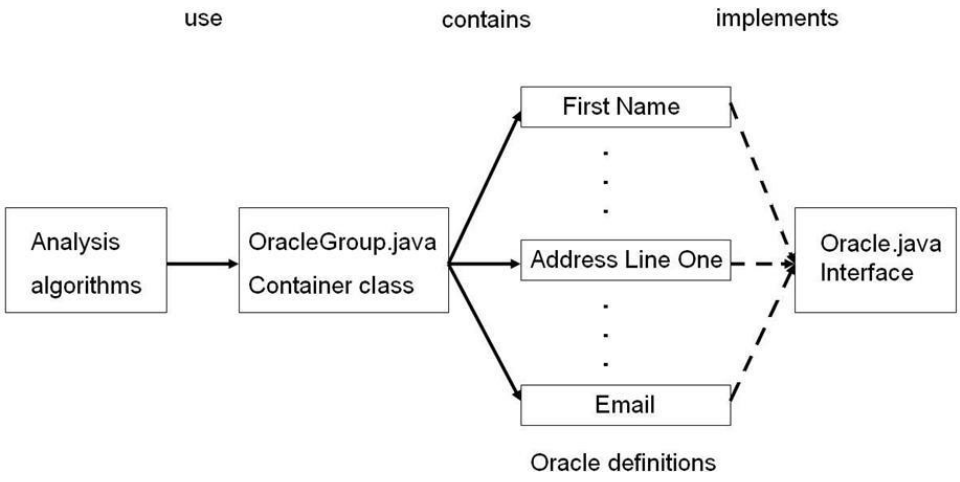

Figure 14: Architectural representation of oracle components.

4.5.1 Interfaces

There are two features that encourage the use of Java interfaces. First, it is preferred that the prototype be extensible, allowing for the inclusion of other content types not currently considered such as social security number or web address. Second, the remaining analysis algorithms have been designed to perform recognition in a generic way thus ignoring any specific instance of an oracle. This approach ensures that the addition or removal of an oracle will have absolutely no affect on the analysis algorithms, decoupling the process of recognition from the tools used for recognition. Together these two design issues do not align with the natural tendency of oracles to be disparate: encouraging special cases and unique logic. The solution is thus to bundle the common functionality that is required by the analysis algorithms into an interface and have each

oracle implement it. A common interface also makes it possible to relate the results from the unique oracles. To this end the Oracle.java interface was defined as given in the following figure.

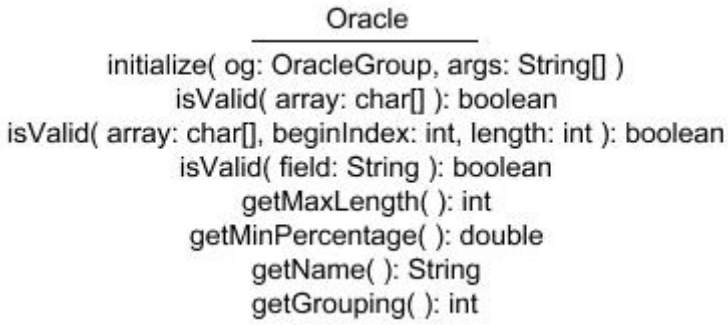

Figure 15: Oracle interface.

The first method is used for initialization where the parameters given in the configuration file are passed as arguments from which the oracle defines its domain. The next three methods (i.e., "isValid()") are used for recognition as they provide access to the domain definition corresponding to the oracle. The three functions provide access to the same supporting logic, but for different input parameter types. The next two methods provide access to instance specific heuristics and thresholds used during recognition while the final two methods provide the associated content type name and grouping respectively.

Certain oracles also implement other applicable interfaces beyond the common interface that each oracle implements. Specifically there are three interfaces that dictate functionality not provided in the basic interface. The first provides the means to direct an oracle to intentionally restrict its domain definition. This ability can be helpful in gaining additional evidence to partially address the domain overlap problem. The other two interfaces define the methods necessary for vertical, cross-reference, and contextual

analysis. There are two interfaces because vertical analysis is performed before conflict resolution and cross-reference and contextual after thus relying on different pieces of evidence.

In implementing these interfaces, the unique domain definition of each oracle may be accessed in a common way. This feature addresses the general and extensible requirements important to the prototype.

4.5.2 Container Class

To further assist with generality and extensibility, a container class was defined that groups the oracles together into a single data structure allowing each to be invoked via this single reference point. This object acts as a wrapper around all of the individual oracles and is an intermediary between the analysis algorithms and said oracles. Which oracles are loaded into the container class and how their internal parameters are set can be parameterized within the configuration file making the engine very dynamic. The container class also has the ability to order the oracles in multiple ways by defining several iterators which sequence through the oracles as specified. This allows the analysis algorithms to view the oracles through the lenses of parameters such as confidence or any other pertinent ordering. Therefore, the container class controls access to the oracles and consequently facilitates their use by the analysis algorithms.

4.5.3 Oracle Definitions

As the oracles' primary role is that of recognition, the methodology they employ in this task is reflected in their definition. As mentioned, this methodology is widely varied among the oracles as it is chosen to correspond to the defined domain. Examples

of appropriate domain definitions include: sorted, lookup tables of valid entries for list-based domains, regular expressions or simple grammars for rule-based domains, or a combination of these. Other conceivable methodologies include incorporating learning algorithms such as artificial neural networks or other probability-based approaches such as the HMMs commonly used in IE and NER or ranked lookup tables. A potential problem with these trained systems is that they rely on contextual characteristics of the data along with the data values. Since context is not reliable in file layout inference, these technologies would typically be reduced to training based entirely on the data values which is not an improvement over lookup tables or predefined rules. Worth mentioning are the composite oracles which are the oracles related to composite content types. These oracles are defined by combining multiple simple oracles into some logic and account for the bulk of the simple grammar-based domain definitions. In this context, the composite oracles rely on the recognition capabilities already defined in the comprising simple oracles. Ultimately, the methodology used for recognition should be appropriate to the domain it is attempting to define.

Beyond the specific domain definition, oracles provide other information corresponding to the associated content type. These include the name or label, the reliability or confidence ranking, and any associated enclaves or groupings. The name is a necessary property as it provides identity once recognition is achieved through the available analysis. The reliability of a content type is determined from the domain definition. Crucial to conflict resolution, this property reflects the amount of confidence the prototype may have in a potential field. As mentioned in Section 6.2 *Related Work*, an inclusive domain definition is generally ranked lower than a more exclusive

counterpart. By assigning an empirical value as the confidence, this property provides important evidence used by the prototype to decide among conflicting potential fields where higher ranked content types are chosen over lower ranked ones. Another feature associated with the oracles is the concept of enclaves or groupings. This property reflects natural relationships that might occur among separate content types and indicates when the valid count acquired after reducing the oracles' domains should be used and when the initial count determined during combinatoric analysis is more appropriate. As an example, when considering records containing USA name and address information, a reasonable grouping would be all of the content types that are associated with an address line one field: street number, directional, street name, street suffix, and address line one. When grouped together, only the primary count (i.e., the initial count determined during combinatoric analysis) should be considered. In fact the secondary count can be detrimental. On the other hand, when comparing street name with content types in other groupings such as last name or city, both counts are important to making an accurate decision. These features all combine to describe a content type and are made available to the analysis algorithms via the oracles.

4.5.4 Oracle Implementations

The implemented prototype was developed to recognize USA name and address information. To this end, twenty-two oracles each representing many of the aspects of name and address information have been implemented, see Figure 16.

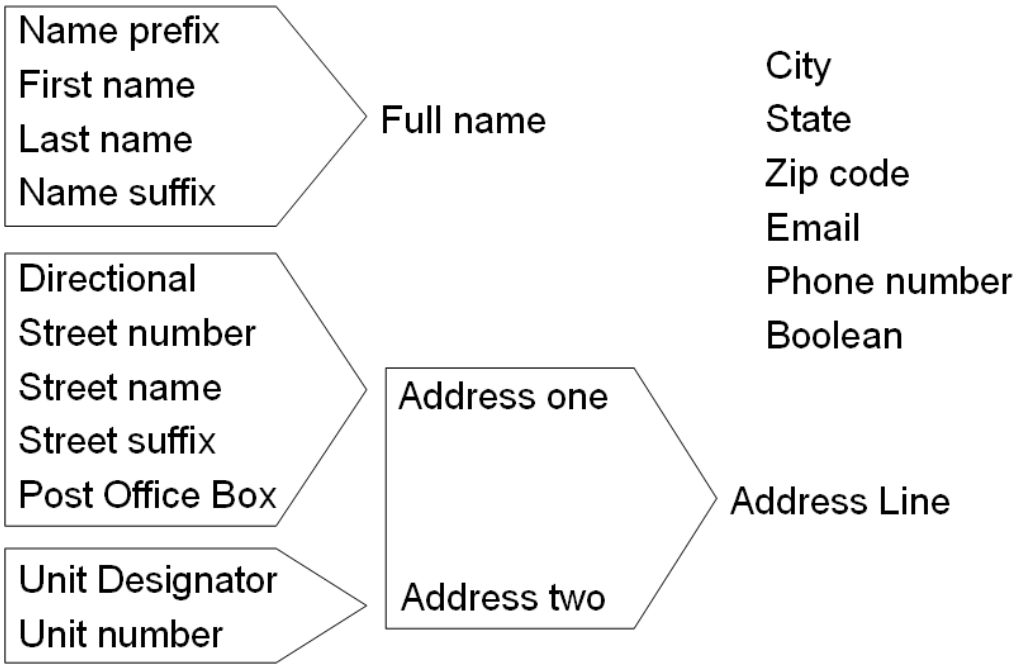

Figure 16: Implemented oracles specified by content type.

4.5.4.1 Name Oracles

There are six oracles representing the various name components often found in records of name and address information: name prefix, first name, middle name, last name, name suffix, and full name. The first five are simple oracles and all but middle name are based on lookup tables of most common entries per census data [40]. Middle name is a unique oracle. It does not actually contain a domain definition as is typical to all the other oracles. Instead it relies entirely on contextual analysis to perform identification. Simply put, a middle name is extremely hard to identify unless it is an initial directly adjacent to a first or last name or a secondary first name relatively close to another name part. Isolated instances can not be reliably classified as middle names and thus context must be the evidence indicating a middle name. The only composite oracle is full name. This oracle combines the previous five into a simple grammar to determine

validity, see Figure 17. Essentially a full name contains at least a first and last name in any order with the other fields considered to be optional. Due to the many legal combinations possible for a full name, this oracle's domain was intentionally defined very loosely.

full-name	optional-parts required-parts | required-parts
optional-parts	optional-parts opt-part | opt-part
opt-part	INITIAL | SUFFIX | PREFIX
required-parts	first-name-one | last-name-one
first-name-one	first-name-one FIRST-NAME | FIRST-NAME optional-parts last-name-two | FIRST-NAME last-name-two
last-name-one	last-name-one LAST-NAME | LAST-NAME optional-parts first-name-two | LAST-NAME first-name-two
first-name-two	FIRST-NAME remaining-parts | FIRST-NAME
last-name-two	LAST-NAME remaining parts | LAST-NAME
remaining-parts	remaining-parts optional-parts | remaining-parts FIRST_NAME | remaining-parts LAST-NAME | optional-parts | FIRST-NAME | LAST-NAME

Figure 17: Simple grammar defining valid full names.

4.5.4.2 Address Oracles

There are thirteen oracles related to address content types. Directional, street number, street name, street suffix, and PO Box are all simple oracles and address line one is a composite oracle associated with the first address line, see Figure 18.

address-line-one	PO-BOX | street-address
street-address	STREET-NUMBER DIRECTIONAL name-part | STREET-NUMBER name-part
name-part	STREET-NAME | STREET-NAME DIRECTIONAL | STREET-NAME STREET-SUFFIX | STREET-NAME DIRECTIONAL STREET-SUFFIX

Figure 18: Simple grammar defining valid address line ones.

Unit designator and unit number are simple oracles and address line two is a composite oracle associated with the second address line. The address line composite oracle

represents a complete address and finally city, state, and zip code are simple oracles representing their titular content types. These oracles are implemented in various ways. The simple oracles are built from lists (e.g., street name, street suffix, zip code) and regular expressions (e.g., street number and PO Box) [41]. The composite oracles are built similar to the full name oracle of the previous section. Worth mentioning is the address line one oracle. Because it is common for a multi-token street name to end with a value that may also be considered a street suffix (e.g., "court") the address line one oracle must be careful how it identifies the individual components within a string of data. To handle this eventuality, it is helpful to find the longest possible street name within the address and only then identify any remaining pieces. This ensures that an address is not preemptively invalidated.

4.5.4.3 Email Oracle

The email oracle is a simple oracle that combines the use of regular expressions with a lookup table. First an input string is compared against a regular expression defined by the RFC specification, see Figure 19 [42]. If the string complies with the defined regular expression then the top-level domain (e.g., .com, .org, and .info) is extracted and compared against a list of valid entries. Due to the expansive nature of the defined regular expression, this assists in determining the correct end position of an email.

addr-spec	local-part "@" domain
local-part	dot-atom / quoted-string
domain	dot-atom / domain-literal
domain-literal	[CFWS] "[" *([FWS] dcontent) [FWS] "]" [CFWS]
dcontent	dtext / quoted-pair
dtext	NO-WS-CTL / ; Non white space controls %d33-90 / ; The rest of the US-ASCII %d94-126 ; characters not including "[", ; "]", or "\"
dot-atom	[CFWS] dot-atom-text [CFWS]
dot-atom-text	1*atext *("." 1*atext)
atext	ALPHA / DIGIT / ; Any character except "!" / "#" / ; controls, SP, and "$" / "%" / ; specials. Used for atoms "&" / "'" / "*" / "+" / "-" / "/" / "=" / "?" / "^" / "_" / "`" / "{" / "|" / "}" / "~"
quoted-string	[CFWS] DQUOTE *([FWS] qcontent) [FWS] DQUOTE [CFWS]
qcontent	qtext / quoted-pair
qtext	NO-WS-CTL / ; Non white space controls %d33 / ; The rest of the US-ASCII %d35-91 / ; characters not including "\" %d93-126 ; or the quote character
quoted-pair	("\" text)
CFWS	*([FWS] comment) (([FWS] comment) / FWS)
FWS	([*WSP CRLF] 1*WSP) ; Folding white space
comment	"(" *([FWS] ccontent) [FWS] ")"
ccontent	ctext / quoted-pair / comment
ctext	NO-WS-CTL / ; Non white space controls %d33-39 / ; The rest of the US-ASCII %d42-91 / ; characters not including "(", %d93-126 ; ")", or "\"
NO-WS-CTL	%d1-8 / ; US-ASCII control characters %d11 / ; that do not include the %d12 / ; carriage return, line feed, %d14-31 / ; and white space characters %d127
text	%d1-9 / ; Characters excluding CR and LF %d11 / %d12 / %d14-127

Figure 19: Grammar defining valid email addresses.

4.5.4.4 Phone Number Oracle

This simple oracle is based on two related lists: one containing area codes and another containing prefixes. For each area code there is a list of valid prefixes. A string to be tested is first checked to ensure that it only contains ten digits. Upon passing this filter it is parsed into its individual components which are compared against the valid lists. First the area code is compared against the corresponding lookup table, if a valid entry is found, the immediate next three digits, the prefix are compared against the list of possible entries for the given area code. The final four digits, the line number, are essentially ignored as they provide little-to-no information and can generally assume any value.

4.5.4.5 Boolean Oracle

This oracle was implemented late as a response to Boolean fields being identified as directional fields (i.e., the value of "N" can also stand for north). The importance associated with this oracle is that it evidenced the positive impact of adding content types to provide additional information.

4.6 Record Length

Returning again to a description of file analysis, from this point forward, the prototype transitions to begin considering the records and the fields they contain. Unlike previously where any analysis was common to all files, the implementation details of the following stages, though addressing a single conceptual problem, will be different for each of the file types as the data must be handled individually for each case. A prime example is the next step in the flow of control: finding the record length. Depending on

the context record length may take on a slightly different meaning, for hybrid or fully fixed files it is the number of characters present in a record whereas in fully delimited files it is the number of fields.

For fully delimited and hybrid files the presence of a record delimiter is an obvious indication of the record length, but in fully fixed files this delimiter clue does not exist and requires other evidence. Originally the name field was chosen as it was thought to be a content type well fitted to the requirements mentioned earlier, and initial tests showed this hypothesis to be accurate. Since record delimiters are absent from the data, the name field will act as a delimiter of sorts by marking the individual records. Though the name field may appear at any position within a record, once located they can still provide enough evidence from which to extract a record length. The first step is to scan the sample identifying the positions where name fields occur.

The record length is determined by examining a range of possible values, selecting the record length that causes the identified positions to line up. Given a range of values, the procedure starts with a minimum value and chooses it as the current record length guess. The data is logically split into records with length equal to the current guess. Using the positional information already gathered, the new records are examined at each position and the number of name fields that occur at the current position is recorded. The field positions line up when enough name fields occur in the same position within a logical record. This value is a threshold that can be set by the user in the configuration file. When this occurs, the current record length guess is assigned as the actual record length. If the fields do not line up, then the guess is incremented and the process repeated.

Once the record length is found, the file is resampled. The new sample extracts data not only from the beginning of the file, but also from equally spaced positions within the file. While this approach can be considered more thorough, it does not guarantee an improvement in the results as the new samples may not boost the random, statistical qualities of the sample. Once extracted from the file the new sample is split up into records to facilitate processing in greater detail. A sample associated with a fully delimited file can be divided into records which may be further broken down into fields, but for fixed files, hybrid and fully fixed, the smallest divisible unit is the record. The fields' positional ambiguity resulting from this discrepancy causes the fixed files to be more difficult to handle than fully delimited files.

4.7 Record Layout Analysis

Since the records have been identified, only a few steps remain, first of which is to identify the fields contained within a record. There are two primary pieces of information to gather when inferring a record's layout: each field's position and type. The field delimiters in a fully delimited file provide all the positional information necessary and as a consequence only the type need be determined. To do this, the records are considered in turn and for each record all of the fields are examined. Each field is passed to each of the oracles, and a count variable corresponding to the particular field and oracle under consideration is incremented if the current oracle indicates that the field is valid with respect to the oracle's domain. Once all of the records have been examined in this manner, the counts are explored to see if any are a large enough percentage of the total number of records to indicate that a field may potentially be of a particular type. This statistical analysis ignores any counts that do not meet the minimum threshold

which is an empirically determined value that can be set by the user and is distinct for each content type. If a count does not meet this threshold, the engine assumes that the entries that were found were in fact false positives due to the fact that only a statistically insignificant set of the records contained fields at the given position that conformed to the associated content type. This methodology is relatively straightforward and is much less involved than that required for the fixed files.

Due to the nature of fixed files, field boundaries are unknown and must be identified along with the field types. Unlike most lexical parsing, which relies on white space and/or punctuation to identify tokens, fields in fixed files may be directly adjacent to each other and as such other clues must be exploited in order to extract any positional information. It is here that the combinatoric approach is used in conjunction with the oracles to provide the necessary clues. In the previous step of the layout inference process, the record length was determined and the data subsequently divided into records. Because the structure is different than that available for fully delimited files the algorithm considers each oracle in turn rather than all at once. For each oracle, the procedure enumerates all start position and length pairs that are possible given the current record length. For each pair the corresponding sequence of characters from each record is compared against the current oracle and a count recorded. Once complete, this combinatoric approach provides all valid counts, as determined by the current oracle, for each start position and length pair. These counts are then passed through a filter to eliminate those that do not meet the minimum threshold. The filter returns a list of potential fields (PF) corresponding to the content type of the current oracle. The details of the methodology employed to build this list is as follows. For each start position, the

valid count at every corresponding length is considered in order to find the maximum valid count associated with the current start position. Once found, this count is compared against the user defined minimum threshold associated with the current oracle to see if the value is statistically significant. If the count is significant, all the adjacent lengths associated with the current start position that have the same valid count are grouped together in a plateau which indicates the full extent of the field. As soon as the edge of the plateau is found, the layout engine assumes it has found the end of the field and adds the length corresponding to the edge of the plateau and its associated start position to the list of PF. As indicated, this process is repeated for each oracle.

Once a list of PF has been acquired a second piece of evidence is gathered: the secondary count. First, each oracle whose domain may be restricted is directed to do so. Once accomplished, the corresponding PF are recounted and the new value stored along with the primary count. This piece of evidence provides information important to the upcoming conflict resolution stage. To indicate this importance some example results from a hybrid file are given in the following figure. The paired bars indicate the primary and secondary counts returned by two oracles, first name on the left and last name on the right, for four related fields: first name, last name, street name, and city name. The dramatic difference among secondary counts shows the value of the secondary count.

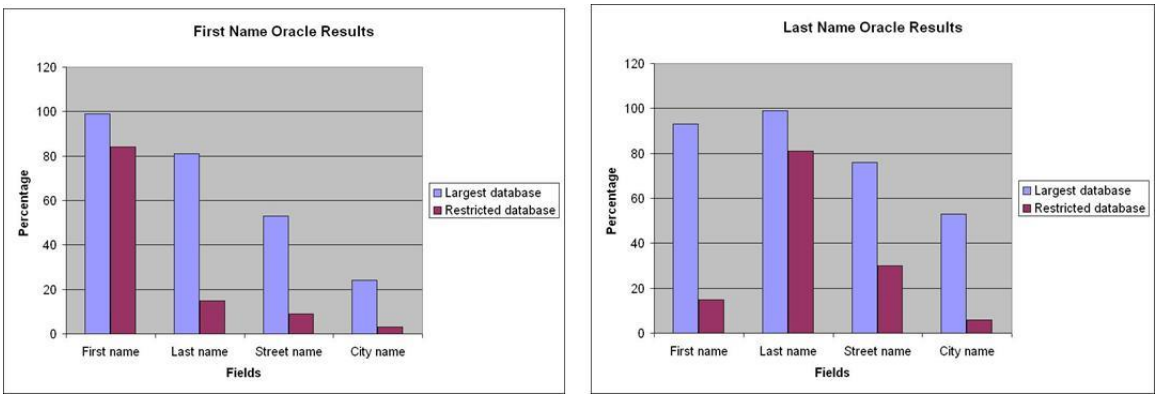

Figure 20: First and last name oracle results for different domain sizes.

4.8 Identifying and Removing False Positives

4.8.1 Vertical Analysis

As the first step in removing false positives, vertical analysis makes use of the individual PF acquired in the previous stage. This is unlike the remaining analysis steps which compare the PF against each other in various ways. In scanning a column of values all-at-once, vertical analysis attempts to recognize any potential data trends that would act as evidence identifying a PF as a false positive.

The process is as follows. Upon instantiation the container class automatically recognizes which oracles implement the interface associated with vertical analysis and groups them together for access by the analysis algorithms. After the combinatoric stage, where recognition occurs, several PF are identified indicating the position and content type of a potential field (i.e., the process just described). Using the iterator provided by the container class any related oracles are sequenced through passing all associated PF to the oracles' vertical analysis functionality. For example, if oracle O implements vertical analysis functionality then all PF of content type equal to the content type of O will be

passed to O for further consideration. In performing analysis O will consider the set of data values corresponding to positional properties of each of the potentially multiple PF in turn. In scanning a set of values, distinctive data trends, refer again to Section 3.3.3.1, that indicate a false positive will be searched for. These are trends that do not conform to the assumption of normal data distribution. If these trends are found to exist the PF is removed from the list of PF, or discarded.

4.8.2 Conflict Resolution

The next analysis step removes false positives by removing any inconsistencies or ambiguity among conflicting PF. This process of conflict resolution can be compared to putting together the pieces of a puzzle. Each content type is considered in turn and each associated potential field is considered a best fit for the corresponding record position if it is better than any potential fields already assigned to that space. The potential field of an oracle is determined to be a better fit if either its domain has a higher confidence ranking or when the ranking is the same the valid count is greater than of any other candidate (i.e. if oracle *A* returns 'yes' more times than oracle *B* for the given position, then *A* is a better fit). As previously indicated, this ranking conveys the property that the domain definition of some oracles is more reliable than that of others. This decision process, while potentially naïve, provides a quick and generic means by which to determine the structure of a record.

4.8.3 Comparison Analysis

Once the conflict resolution stage creates an initial representation of a record, the remaining analysis attempts to further improve the accuracy of the result by comparing

fields assigned to the record against each other. Two implemented approaches include cross-reference and context analysis. Cross-referencing compares the data values associated with related fields (e.g. city, state, and zip code) and removes any entries assigned to the record structure for which there is no match (e.g. an extra zip code). Context analysis uses the relative nearness of fields of similar content type to identify false positives. Where cross-reference is based on the data values contained within the fields, context relies on the content type associated with the fields. To reiterate, context is unreliable when applied to the data file layout inference problem and must be used with great care. Typically, this form of comparison is best employed to remove PF associated with content types defined by very inclusive domains by only retaining those PF that are relatively near PF of similar yet more reliable content types. Another reasonable scenario is to identify otherwise unidentifiable fields such as a middle initial. Together, these comparison techniques provide more evidence to identify and remove false positives.

The analysis performed after the combinatoric stage can provide significantly better results with relatively little computational cost. As with vertical analysis it is important to preserve the architectural concepts of generality and extensibility and thus the details of the cross-reference and context functionality must be built into the oracles themselves and accessed via methods of an implemented interface. Oracle designers can choose whether or not to implement this interface and consequently only the oracles that implement this interface may be accessed through the previously mentioned container class.

4.9 Header Record

The final processing stage only applies to fully delimited files. In this scenario, it is possible that the first record might be a header record which contains structural information associated with the records of the file. It is important to detect such a record. Using the results of the decision stage, the first record is examined to see how many of the fields have values whose respective types match those indicated by the determined record structure. If a statistically significant number of fields have corresponding types, then the record is not assumed to be a header record, otherwise it is. Future work will be to determine how best to exploit any available information in this record to improve the results or even avoid the processing steps altogether [38]. The wide variety of possible content type labels seems to be prohibitive in this regard.

4.10 Output

Finally, the engine has completed its analysis and is ready to report its findings. To do so all the evidence recorded so far is bundled into either formatted text or XML, an option that can be specified by the user, and written to an output file. This file reports runtime, the existence of a header record, what delimiters are present, what oracles were considered, the record length, the record structure, and all PFP that passed their respective minimum thresholds. This output is the final result of the layout inference engine prototype, examples of which may be found in Appendix B.

5. RESULTS

5.1 Results Description

Given the thesis of this work, the prototype was incrementally enhanced, exploring new issues as they were encountered, in order to provide continually improved accuracy. Initial tests involved a collection of synthetic data files generated by the University of Arkansas' Synthetic Data Generation program [43]. These files served as a proof of concept with respect to the layout engine, but were quickly outgrown as real data files became available. The synthetic files usefulness was restricted because, among other issues, they were much too clean, did not contain multiple occurrences of field types, and rarely contained sparse fields. Consequently their ability to expose issues encountered within real data files was limited. The test suite of real data files was supplied by an industrial sponsor in two stages: first a collection of six files and then another collection of seven files. Excluding proof of concept, these thirteen files have served to indicate the prototype's performance both with respect to accuracy and runtime.

The prototype's development has been driven by evaluation and understanding of these acquired results and has followed an iterative pattern. First, the prototype and its supporting theory are evaluated against existing results to determine what enhancements might provide improvements. Next, the prototype is modified to include the suggested enhancements. Once completed, the results of the updated prototype are compared against pre-existing results in order to determine if an improvement is achieved. This comparison provides evidence for or against the suggested modifications. Based on this information, the prototype is brought to a corresponding stable state and the entire process is repeated. This approach is similar to test driven development where expected

results drive the development process [44, 45]. This approach to development has provided the impetus for developing much of the analysis performed outside of the combinatorics stage.

This type of development has the downside of potentially being tuned to the test data, particularly for data analysis processes such as this. In an attempt to limit the affects of over-tuning, the proposed theory has been extrapolated to address the encountered issues in a generic way so that, when related issues are encountered within unfamiliar data sets, the prototype can perform correctly. As mentioned in Section 4.5.1, this is also a design goal for the prototype. For example when fields were discovered that contained a single data item for each non-blank entry this issue was identified as a part of a greater issue of limited statistical distribution. Subsequently vertical analysis was introduced to identify and address limited statistical distribution. In this way a single instance is used to identify and solve more generic issues.

5.2 Results by File

The following sub-sections each provide a brief description of a single file, and the corresponding prototype results for each of the thirteen files. These sub-sections are followed by a results summary which will present aggregate results for all the files. There are two charts presented with each file. The first chart depicts the accuracy of the prototype for varying record sample sizes. One form of sensitivity analysis, this chart shows how accuracy can be affected by different sampling sizes. Again, it is important to note that the prototype samples multiple, equally spaced sections from a file and thus some or none of the records in a sample of 50 records may be included in a sample of 150

records. For this chart there are four series representing four different counts: correct, false positive, missed, and total.

- A correct field is a field that has an accurately assigned position and type according to a provided file schema.
- A false positive field is a field that has been incorrectly assigned by the prototype to the record schema. This constitutes a field of unknown or unrecognizable type being assigned a type identifiable by the prototype, also see 3.3.2.
- A missed field is a recognizable field (i.e., a field whose corresponding type is defined within the prototype) that is either mislabeled or not labeled at all.
- The total count is the sum of recognizable fields within the record schema. Added together the correct and missed counts equal the total count.

Besides accuracy performance, runtime has also been recorded in a second chart. Due to the nature of Java, especially with respect to the garbage collector, often a single runtime is not sufficient to determine the representative runtime. For the results reported here, the prototype was timed on multiple runs for each record count for each file. A single value was determined by discarding any extreme outliers and averaging the remaining values.

5.2.1 Synthetic Files

As mentioned, synthetic files were the initial test bed against which the prototype was run. Serving as a proof of concept, the results associated with these files served to indicate that the implemented approach was both feasible and desirable. Sensitivity analysis was not performed on these files as the results from the real data files are

considered to be a better representation. Each synthetic file contains a total of 2000 records and represent both fully delimited and hybrid file types. Though not reported here, several hybrid files were stripped of their record delimiters transforming them into fully fixed files. The results were unaffected.

As seen, the prototype performs well against the synthetic files. Since a few of the files contain an isolated middle name initial, the results could be considered to be better than they appear. Particularly in a hybrid file, this situation is extremely difficult to identify even by a human. Other missed fields were primarily a result of extremely sparse populations. An example of this would be address line two components where only a couple of fields contain values among the total sample. This is partly what caused the poor results in files eight and nine, files which actually consist of the same data but represented as different file types. Another issue is that the synthetic files showed poor statistical distribution. Consequently, the issue of domain overlap is exaggerated, making the conflict resolution stage difficult. In the following chart, the total length of the bars represents the total number of fields to be recognized for a corresponding file. This bar is divided into correct and missed field counts as indicated by the legend.

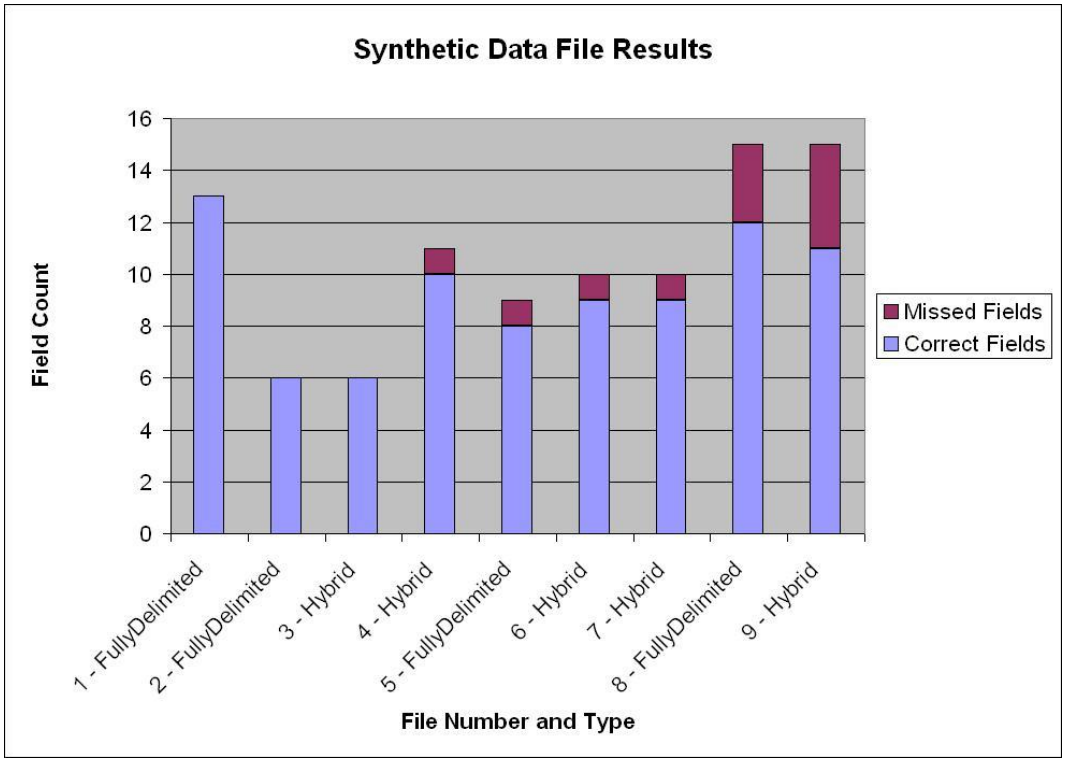

Figure 21: Results for nine synthetic data files.

5.2.2 File 1

The first file is a hybrid file containing 100 records with record length 177 and thus the results do not change after the number of records examined reaches 100. The prototype performs quite well for this file. The records in files one, two, five, and six are comprised entirely of content types recognized by the prototype: name, address, city, state, zip code, phone number, and Boolean.

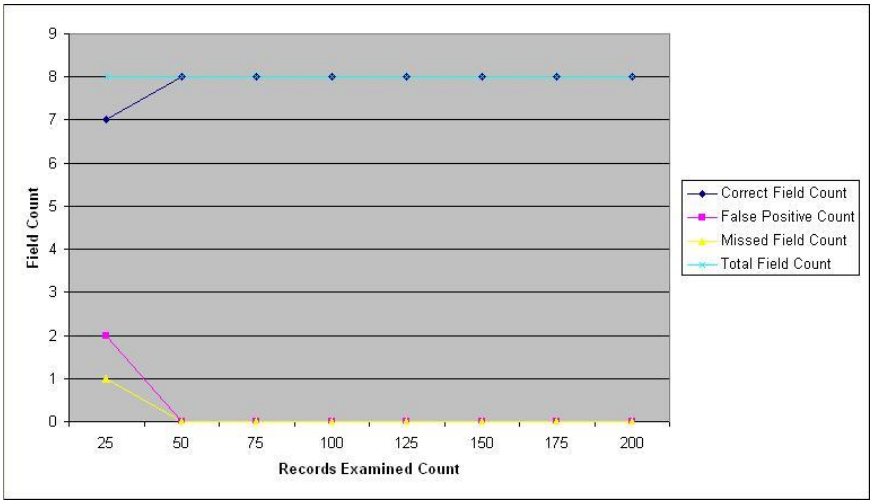

Figure 22: Accuracy performance for file 1.

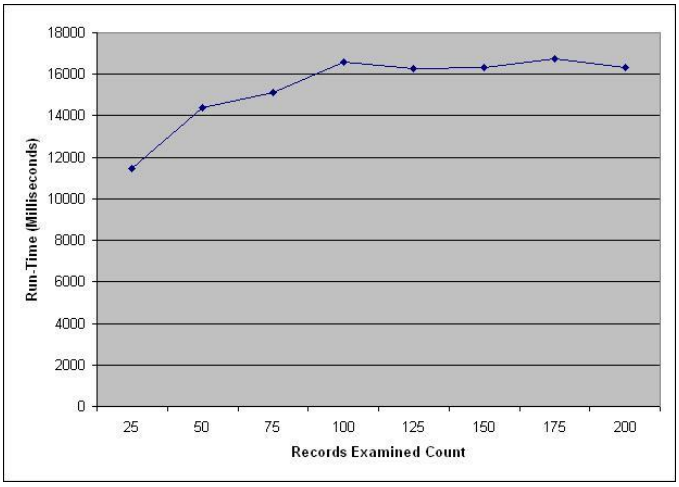

Figure 23: Runtime performance for file 1.

5.2.3 File 2

File 2 is also a hybrid file but with record length 186. This file contains a total of 100 records and thus the results do not change after this point. The prototype performs quite well for this file.

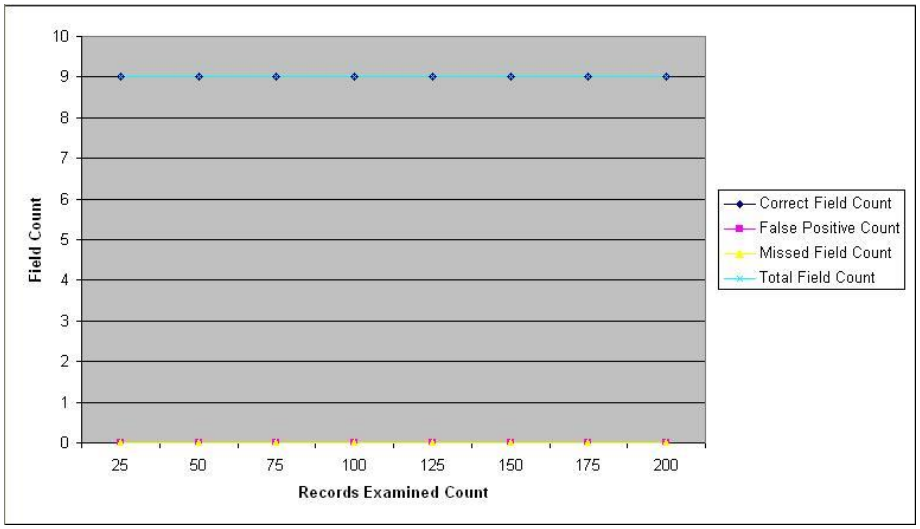

Figure 24: Accuracy performance for file 2.

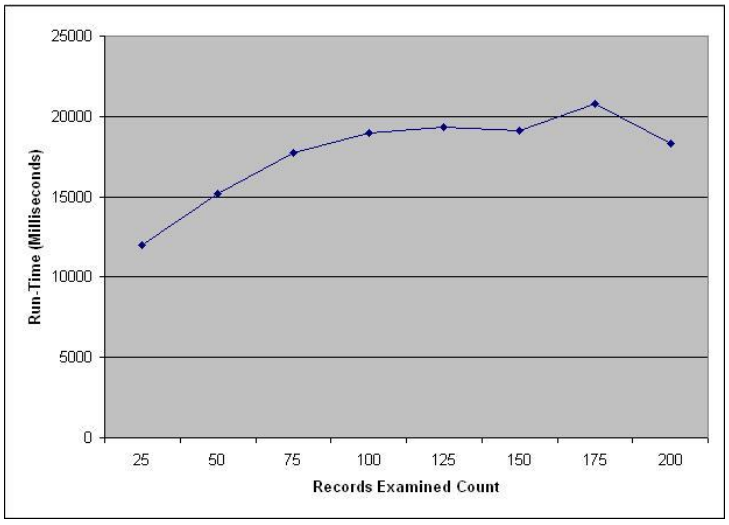

Figure 25: Runtime performance for file 2.

5.2.4 File 3

This file is a fully delimited file with 89 fields. Notice the significant improvements in the runtime over hybrid and fully fixed files. This file contains 100 records and thus the results do not change after this point. File 3 is the only fully delimited file considered for the real data files. With 89 fields in a record, this file contains repeated fields, sparse fields, composite fields broken into their simple parts, and isolated fields all representative of situations encountered in real data files.

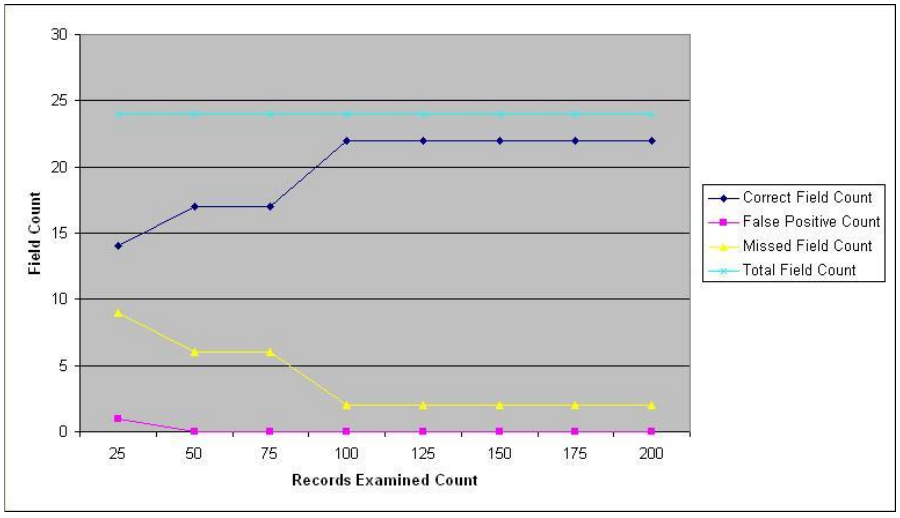

Figure 26: Accuracy performance for file 3.

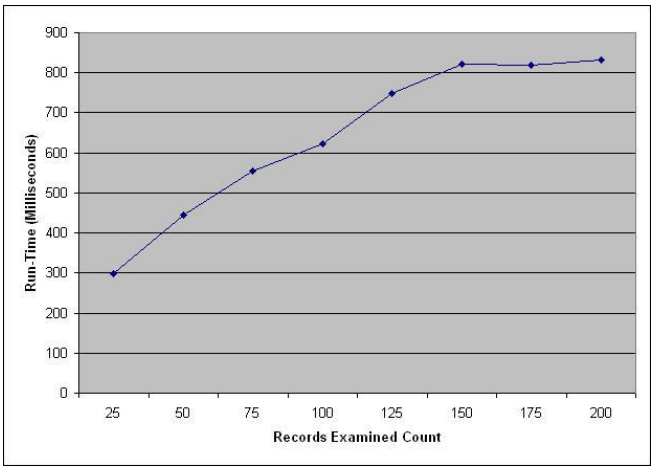

Figure 27: Runtime performance for file 3.

5.2.5 File 4

File 4 is a hybrid file with record length 681. This file contains a total of 100 records and thus the results do not change after this point.

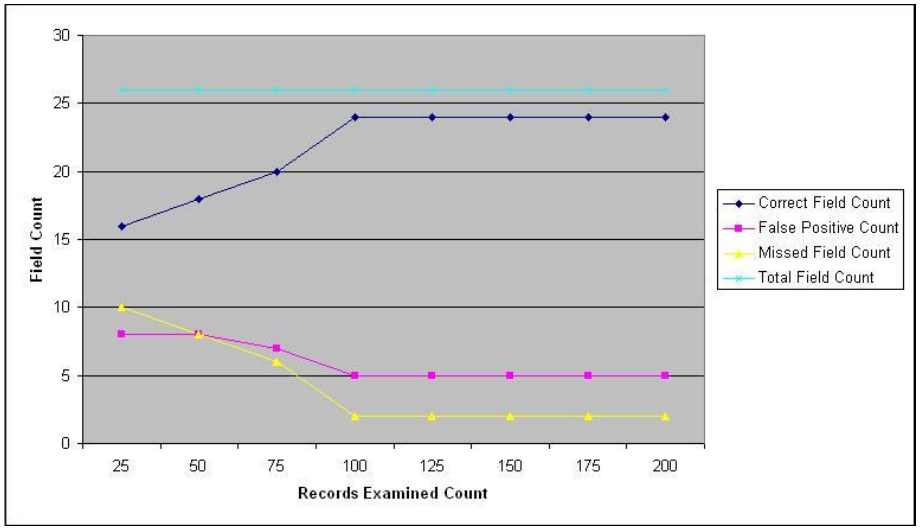

Figure 28: Accuracy performance for file 4.

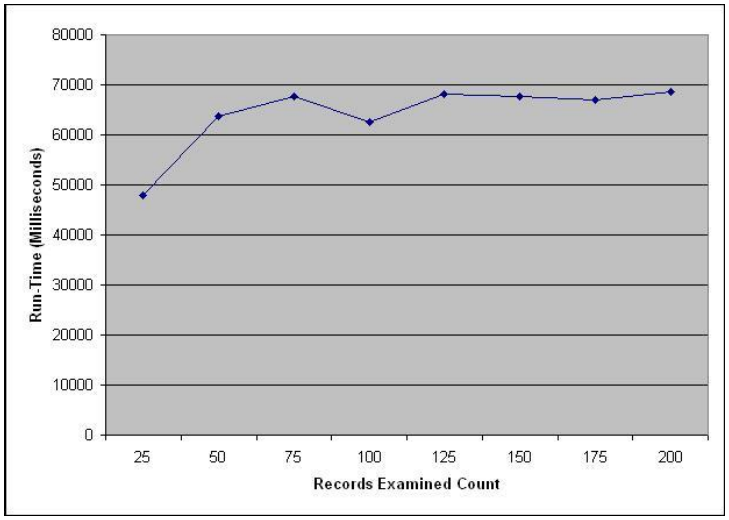

Figure 29: Runtime performance for file 4.

5.2.6 File 5

File 5 is a hybrid file with record length 182. This file contains a total of 100 records and thus the results do not change after this point.

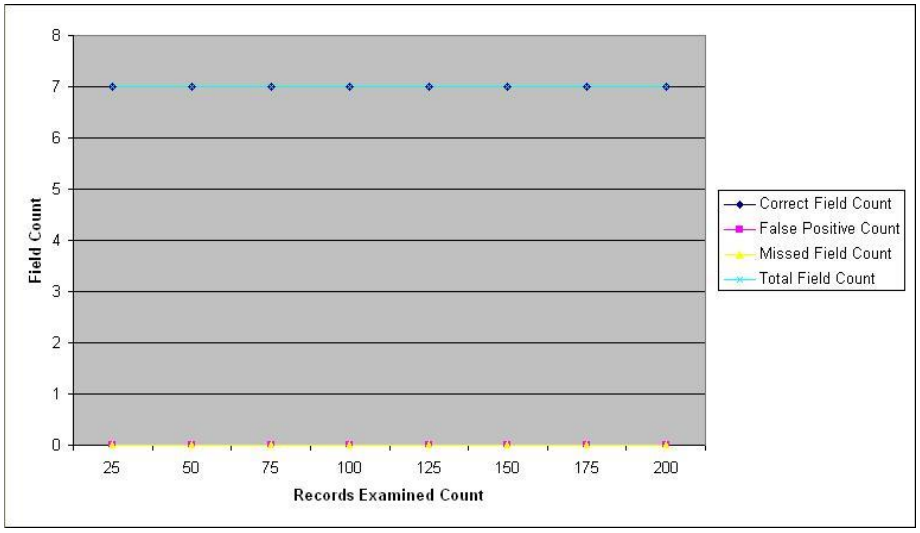

Figure 30: Accuracy performance for file 5.

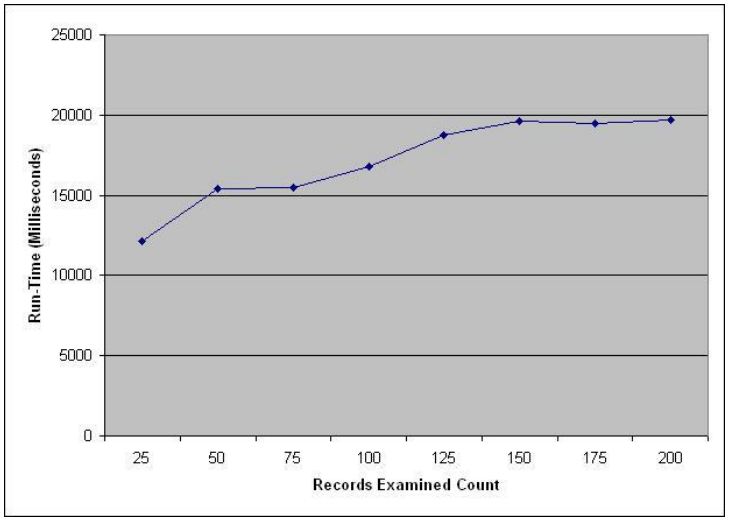

Figure 31: Runtime performance for file 5.

5.2.7 File 6

File 6 is a hybrid file with record length 133 containing a total of 225 total records. The results spike at 125 and 150 records due to the values being sampled. In these two instances only, a sparsely populated address line two field is not blank and is recognized. For the other sample sizes the field contains only empty values and is not recognized. Consequently the separate address components, line one and two, are recognized as a complete address line field.

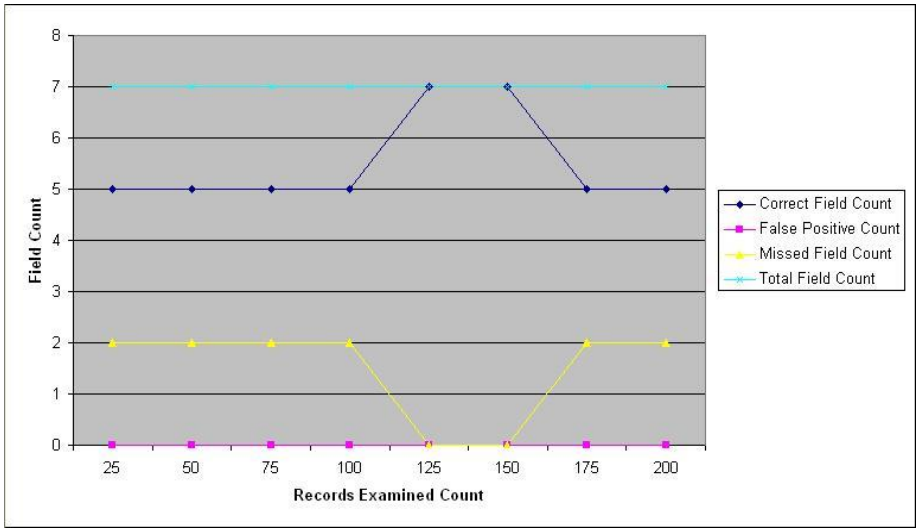

Figure 32: Accuracy performance for file 6.

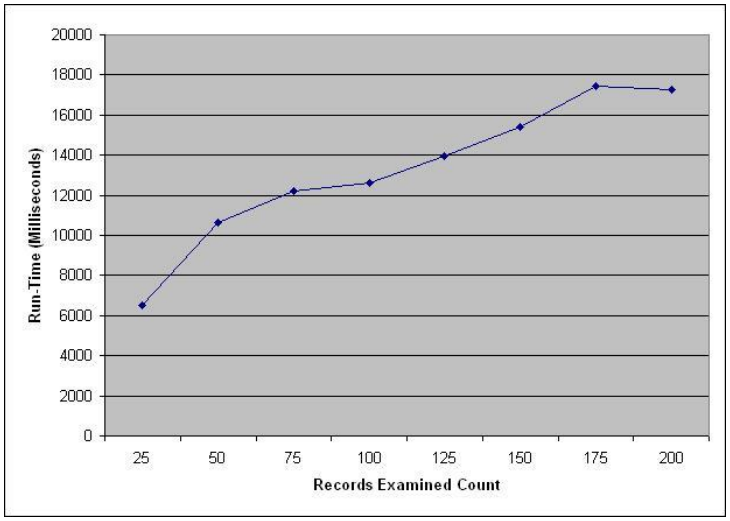

Figure 33: Runtime performance for file 6.

5.2.8 File 7

File 7 is a hybrid file with record length 845. This file is the worst performing file encountered thus far. This poor performance is caused by several sparse fields, fields containing mostly blank entries. These fields are historical addresses that are inapplicable for most records and thus left blank. Consequently the statistical assumptions defining this approach are not met and the fields are not reported.

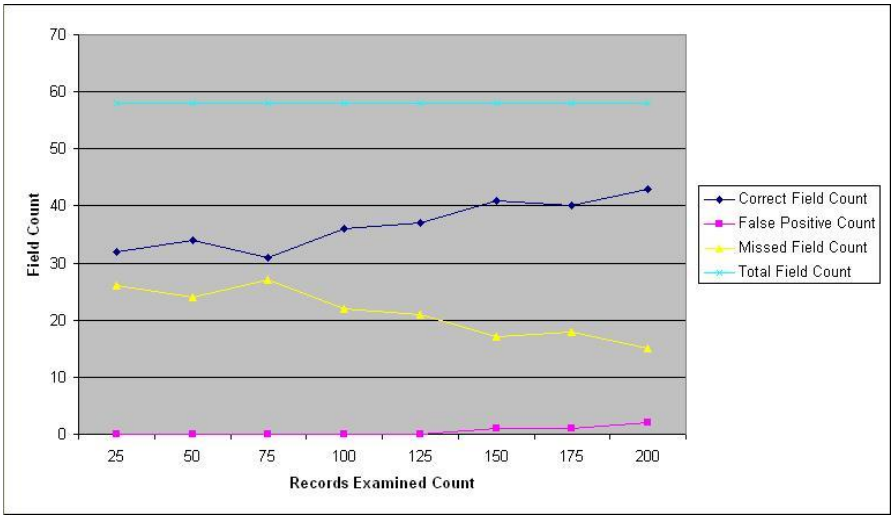

Figure 34: Accuracy performance for file 7.

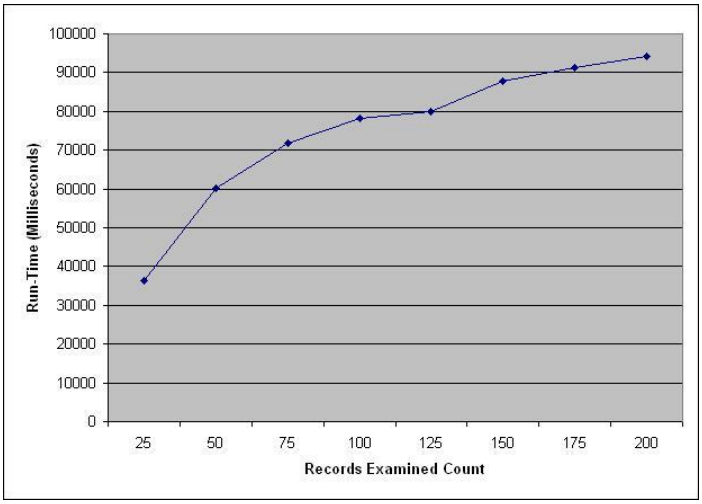

Figure 35: Runtime performance for file 7.

5.2.9 File 8

File 8 is a fully fixed file with record length 368.

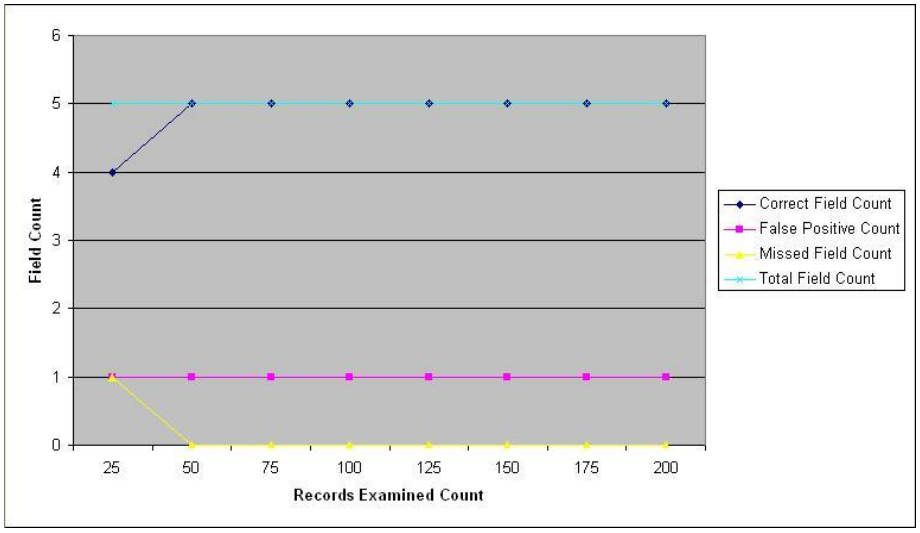

Figure 36: Accuracy performance for file 8.

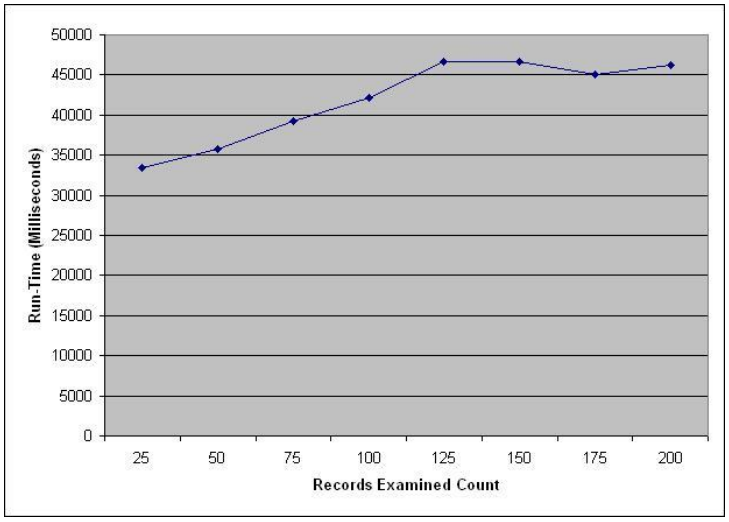

Figure 37: Runtime performance for file 8.

5.2.10 File 9

File 9 is a hybrid file with record length 1141. This file contains a longer record length resulting in significantly longer runtimes, a feature discussed in the Runtime

section. The number of false positives is a result of sparse, yet formatted data (e.g., values that contain only a 'Y' or 'N' at a given position within the field which can be misinterpreted as a Boolean). The fact that many of these fields are sparse further complicates the situation, limiting any discrediting information.

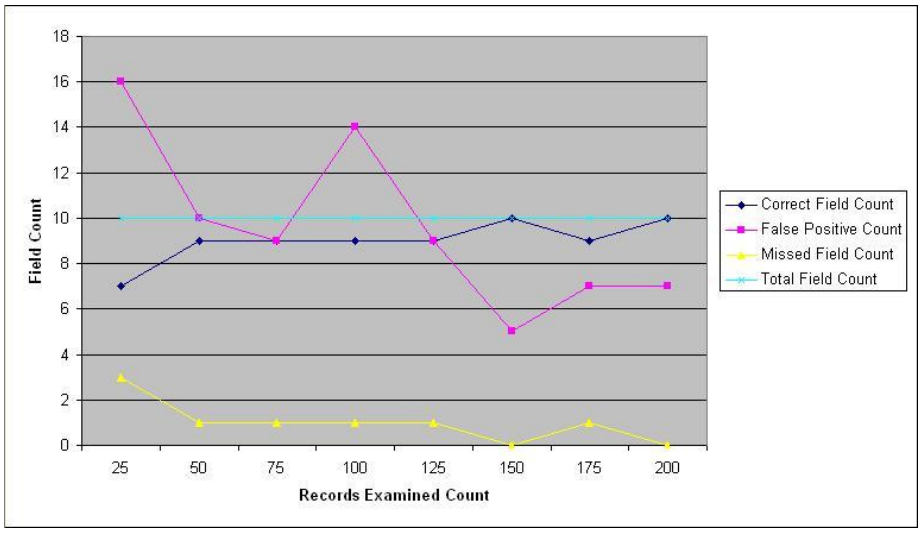

Figure 38: Accuracy performance for file 9.

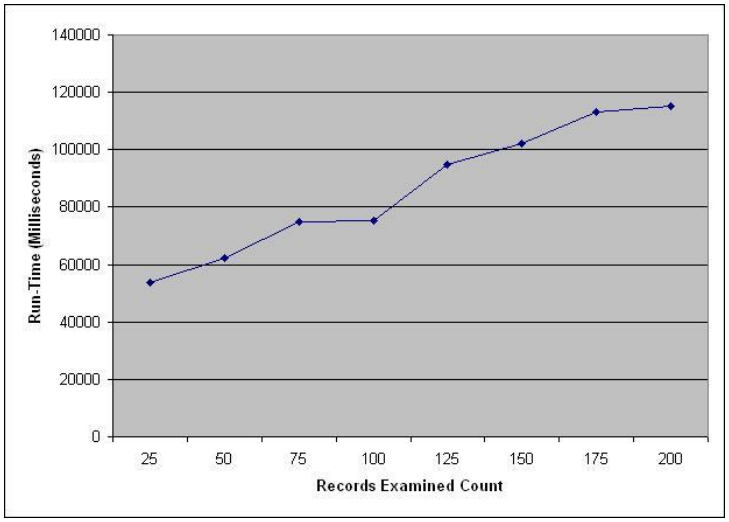

Figure 39: Runtime performance for file 9.

5.2.11 File 10

File 10 is a fully fixed file with record length 1053. This file suffers from a similar issue experienced with file 6. In this file a duplicated address line one field is only partially recognized as several entries contain information extraneous to the oracle's domain definition. Thus, just as in file 6, changing the sample size and consequently the records that are sampled affects the results.

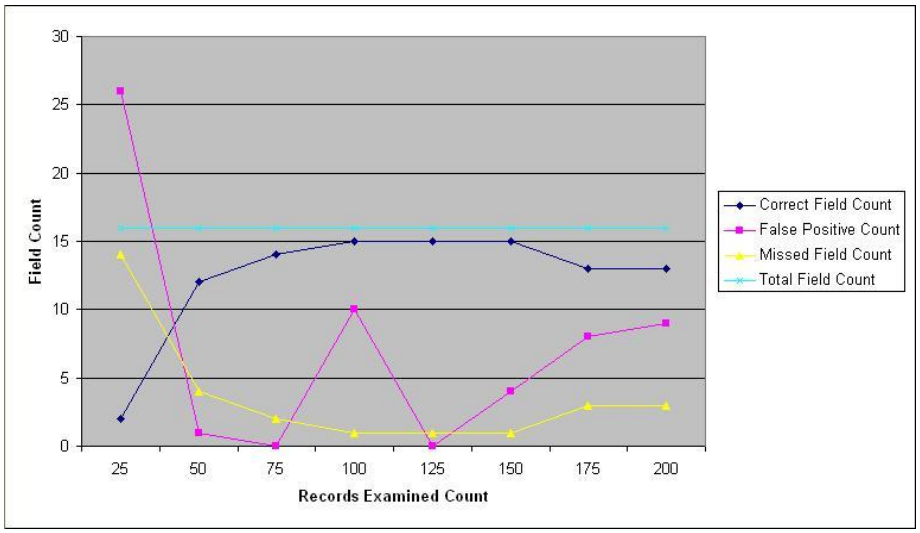

Figure 40: Accuracy performance for file 10.

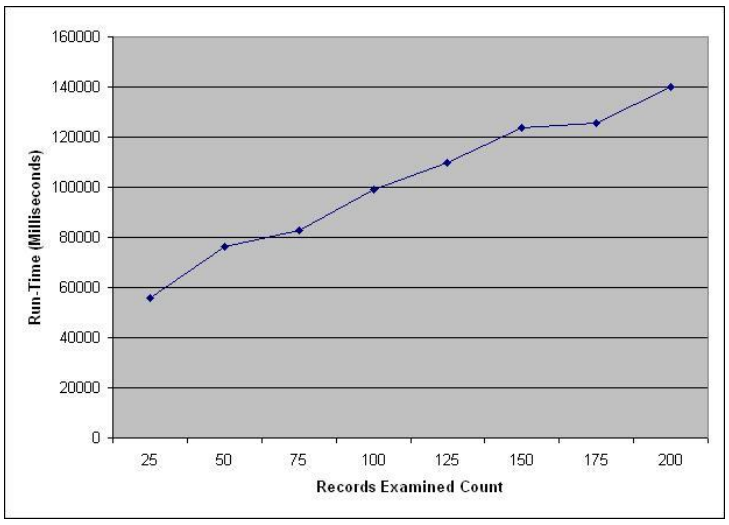

Figure 41: Runtime performance for file 10.

5.2.12 File 11

File 11 is a hybrid file with record length 2689. In this file an address line one field is only partially recognized. Analyzing the sampled data indicates that a better domain definition would immediately improve the results.

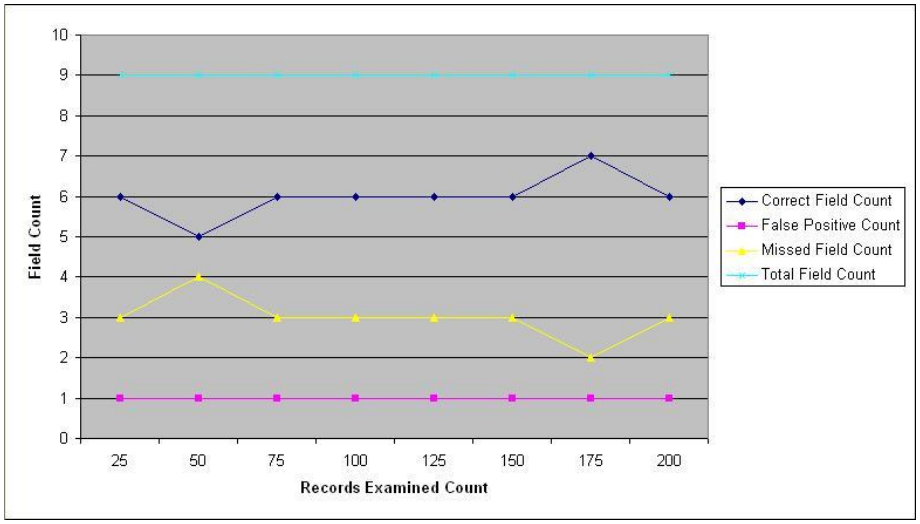

Figure 42: Accuracy performance for file 11.

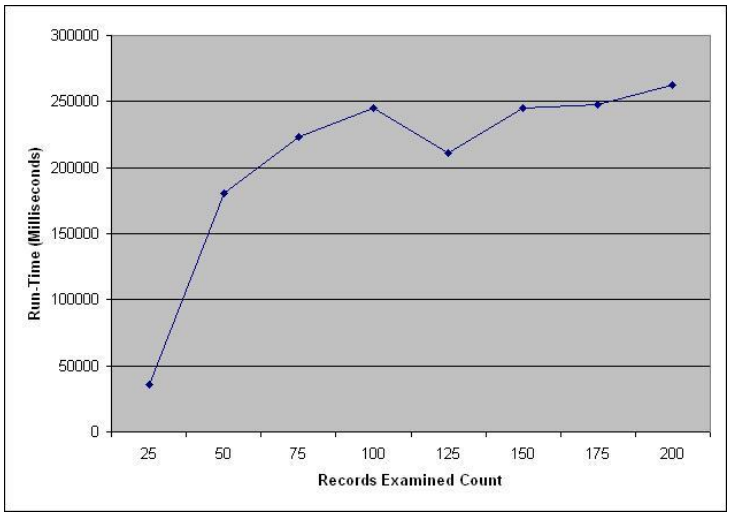

Figure 43: Runtime performance for file 11.

5.2.13 File 12

File 12 is a hybrid file with record length 1886.

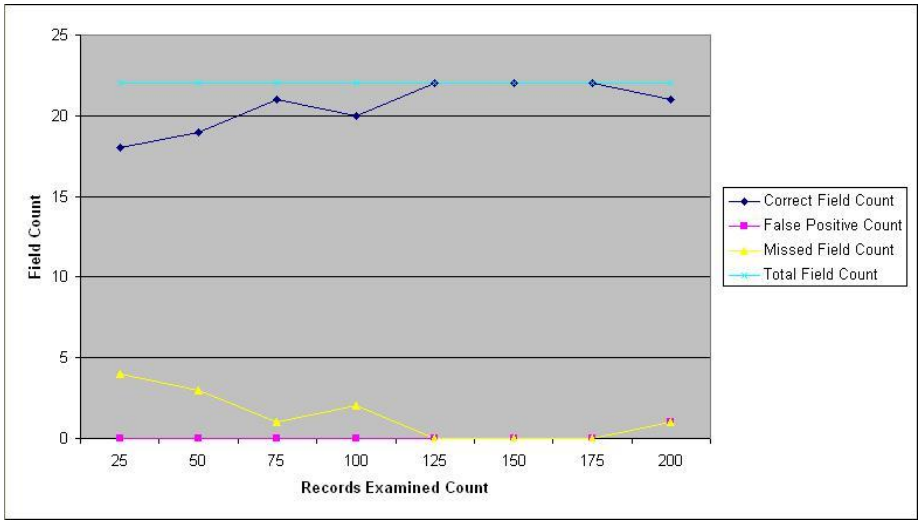

Figure 44: Accuracy performance for file 12.

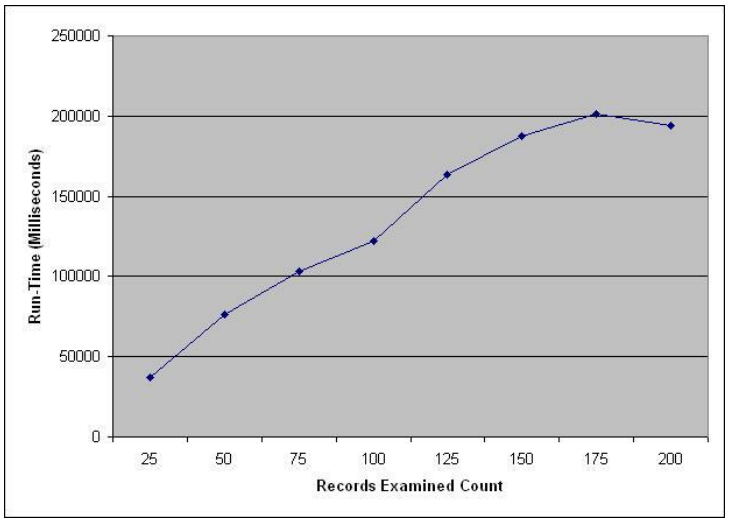

Figure 45: Runtime performance for file 12.

5.2.14 File 13

File 13 is a hybrid file with record length 1886. The results of this file show the importance of comparison analysis. The false positive count is increased for this file

because a city field is misidentified. This field is missed because all entries are the same, a value shared by the last name oracle, which is chosen during conflict resolution. The problem of no statistical distribution creates a scenario where the gathered evidence is inconclusive, providing no indication which PF to choose. Consequently the corresponding cross-reference logic is unable to complete its task removing other related fields.

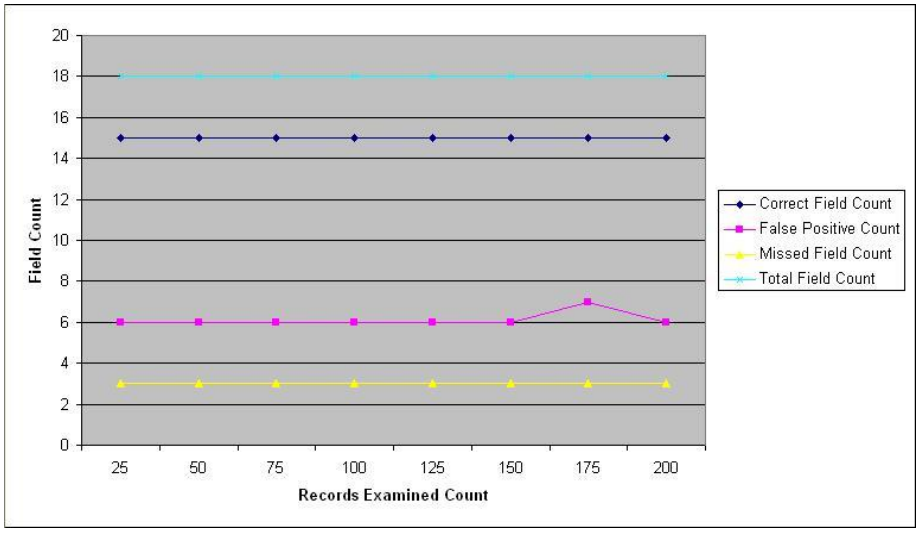

Figure 46: Accuracy performance for file 13.

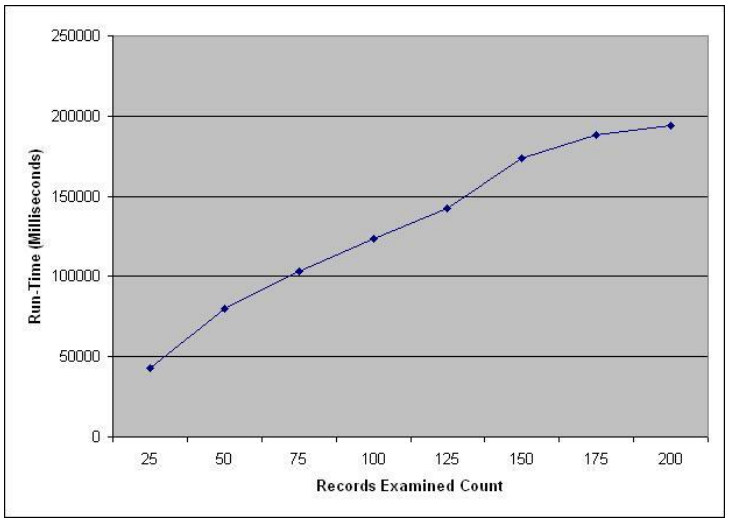

Figure 47: Runtime performance for file 13.

5.2.15 Summary Statistics

The following figures give summary statistics for the previous files. Indicating identical trends, the top two charts provide summation results while the bottom two give an average. Summation results are a sum of the respective counts, or runtimes, compared against other sums. Average results are the average of the respective counts, or runtimes, compared against each other. Figures 48 and 50 show accuracy performance of the prototype, and Figures 49 and 51 indicate runtime performance of the prototype.

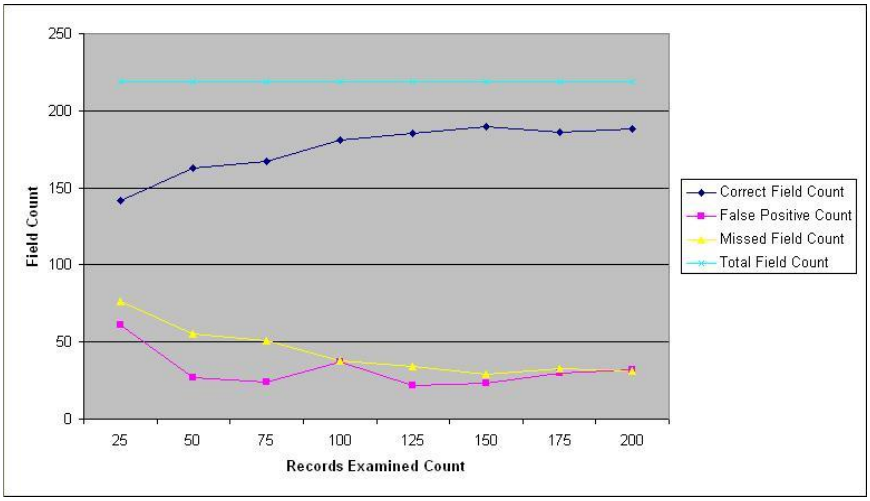

Figure 48: Accuracy performance summary - sum.

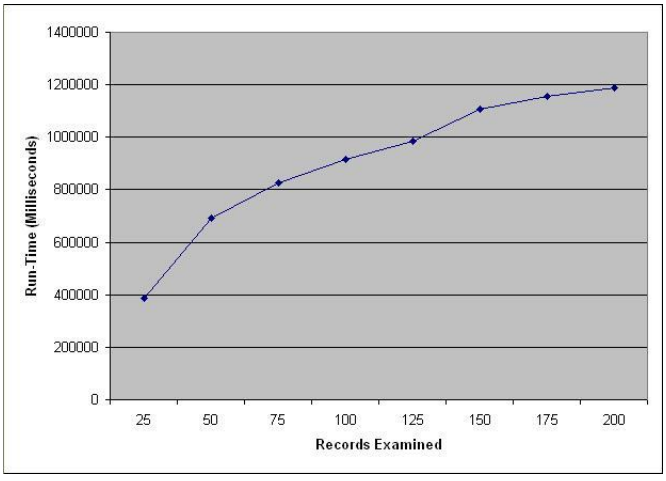

Figure 49: Runtime performance summary – sum.

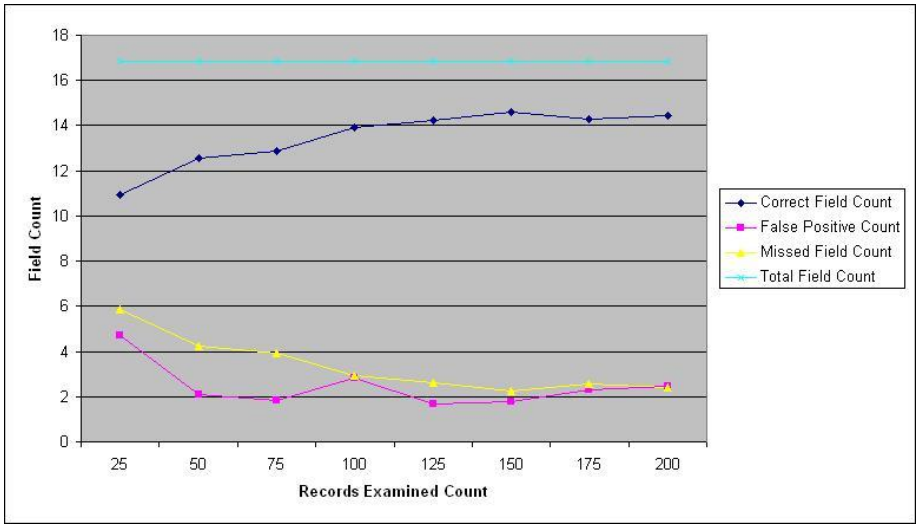

Figure 50: Accuracy performance summary - average.

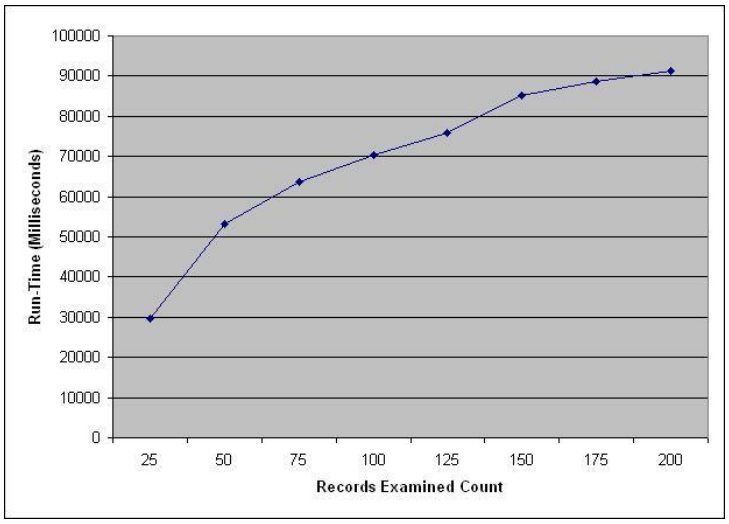

Figure 51: Runtime performance summary - average.

The following figure compares correct and missed field counts for each of the real data files which are indicated by number and type. These are the results for 150 records, the number of records for which the prototype consistently performed best. This figure relates to the chart provided for the synthetic data files. As is indicated in the chart, the many sparse fields in file seven provides an area for improvement.

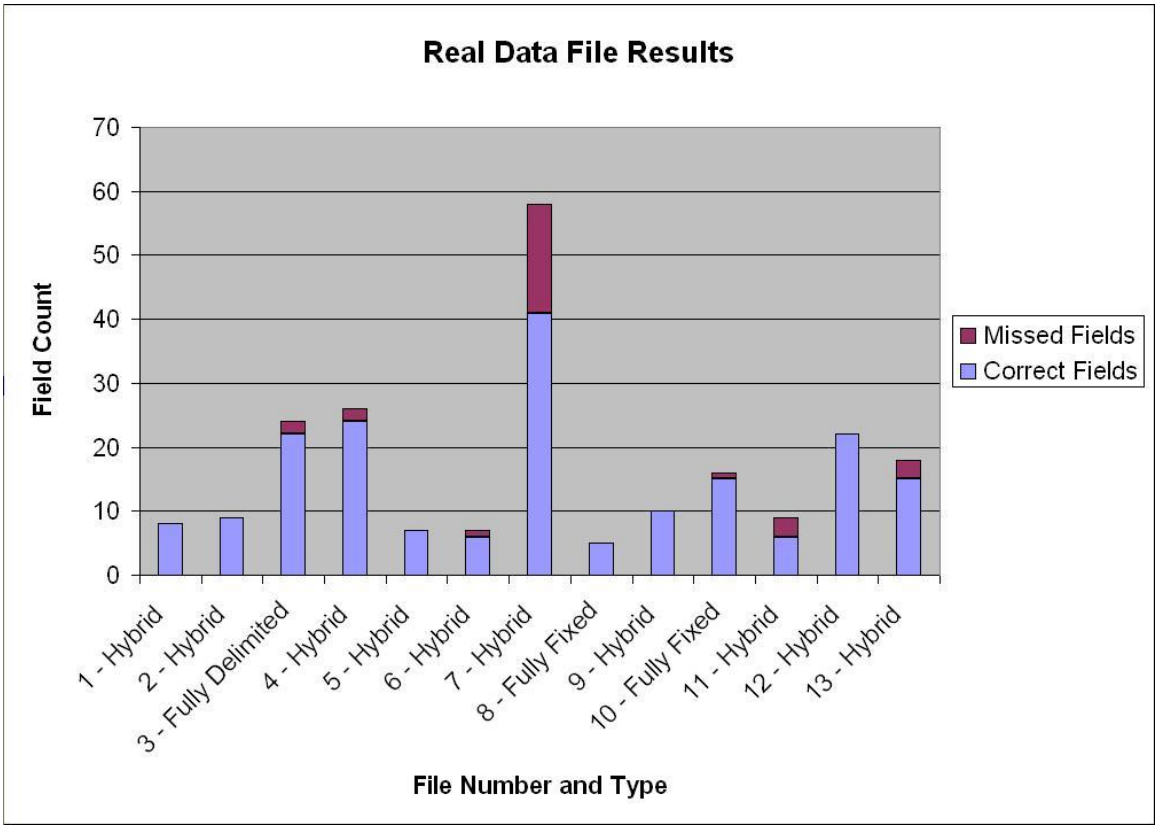

Figure 52: Results for thirteen real data files.

5.3 Runtime

Due to the highly configurable nature of the prototype, it is difficult to propose an encompassing runtime analysis in an understandable and concise manner. What can be provided are the two factors that determine the runtime's theoretical upper bound: the record length and the number of records examined. The most computationally intensive step is the identification stage when PF are located using a combinatoric approach that considers the sequence of characters from all the available records corresponding to each possible start position and length. This indicates that the number of sampled records multiplied by the square of the record length is the computational upper bound of the layout engine. Of course, this is different than the actual bound due to implemented

heuristics described in Section 4.1. The runtimes provided for each of the thirteen files indicate that the run time is primarily influenced by record length and secondarily by the number of records. Timings for other computational steps, with the exception of finding record length in fully fixed files, require immeasurably small amounts of time to complete.

6. CLOSING REMARKS

6.1 Conclusions

As defined in the introduction, data file layout inference is the process of determining the organizational properties and characteristics, or layout, associated with a data file. Character encoding, delimiting characters, file type, and record structure are all important features of data file layouts as they must be known if any data is to be extracted from a file. Within this paper, an approach has been defined for the automatic detection of these properties. This suggested approach has been implemented as a working prototype from which results have been obtained. Beyond implementing the proposed methodologies for recognition of the various layout characteristics, the layout engine has also been architected in a general and extensible manner. This provides an extensible way to add or change various components of the prototype (e.g., additional content types, other forms of evidence, and new methods for analysis). The accuracy results returned by the layout engine indicate the correctness of the suggested approach. Overall, the results are accurate more than 85% of the time which is better than or equal to the results of the IE and NER systems which provide the most comparable technologies. For primary fields, fields that are not sparse or that do not contain extraneous information, the results are significantly better as evidenced by the results for files 1 and 2, among others.

The impact of non-primary fields on the results suggests the importance of the assumptions presented in the introduction. Of particular note is the second assumption which requires that errors (i.e., unrecognizable data) represent a statistically insignificant portion of the data. When a field is sparse, mostly blank entries, each unrecognizable entry makes it much more difficult to accurately assign a content type to the

corresponding field. Taken further, this suggests that the approach described for the layout inference problem may not return the correct layout all of the time. There are files that do not conform to the statistical assumptions made by the given approach, data files that even humans might struggle with, that the layout engine cannot handle. An example of which could be records with isolated middle initials or number fields such as a phone number or zip code among several other number fields.

6.2 Future Work

While the described approach has been shown to be appropriate, even providing encouraging results, for the layout inference problem, certain aspects remain to be explored. From relatively minor expansions on existing logic to drastic revisions of the approach there are multiple possibilities to consider that could improve performance or expand capabilities.

6.2.1 Content Types

One candidate for future work involves content types and their associated domain definitions. The oracles created for the existing prototype only recognize USA personal name and address type fields. This set can be expanded to include other content types and other nationalities.

Expanding the set to include other content types might introduce another sub problem, specifically identification of the type of data stored in the file. Beyond recognition of individual fields, a more general data classification might be possible (e.g., a file containing records of purchase orders or business names and addresses).

Internationalization is another challenge worth considering. By decoupling the recognition features provided by the oracles from the analysis algorithms, it is expected that such a transition would be a straightforward process with the only modifications to be to define and add new oracles.

Beyond expanding the domain of the layout engine, another benefit of additional content types is that they can positively affect accuracy. This is evidenced by improved prototype results from the inclusion of more oracles (e.g., when the Boolean oracle was added some directional false positives were updated to the correct type). This is a direct consequence of the extra evidence provided by the new oracle (i.e., domain definition, oracle ranking, and any applicable analysis steps). Once the recognition functionality is properly defined and generated, any necessary oracles can be easily included in the prototype via the configuration file by specifying the oracle's qualified name along with any associated parameters.

When creating an oracle for a content type, the domain definition is an important component and thus is a topic of continued work. An incomplete domain definition can cause an oracle to under-perform generating an incorrect layout. Using the existing prototype as a reference, this problem can be shown in two specific instances. First was a first name field that included many partial business names. Second was a street name field that included the partial entry "PO Box." While the address line one composite oracle was defined to handle post office boxes, the simple street name oracle was not. These instances provide examples of potentially incomplete domain definitions that if updated may improve the prototype's results.

An unexplored area that could provide significant improvements in accuracy, particularly by better differentiating between correct fields and false positives, is the ranking of domain entries. If each entry has a corresponding ranking associated with it, then the evidence can include a more direct indication of reliability in conjunction with the already acquired counts. This type of evidence seems appropriate given the statistical assumptions dictated by the approach. In fact recording the probabilities of the entries could eliminate the need for many instances of vertical analysis. With this approach the difficulty lies in determining the correct probabilities for each entry. For some content types such as names, available census data makes this process very easy, but for others such as number fields such a ranking may prove to be meaningless if not impossible. The concept of ranked entries is partially included in the prototype, but unintentionally and not explicitly. Certain oracles' domains do not include all possible entries because it is either impossible or unnecessary. In this case the included entries are obviously ranked higher than the excluded entries. Intentionally or necessarily restricting an oracle's domain definition can cause problems though and should not be considered a direct substitute to individual domain entry ranking.

Specific to the prototype, both the combinatorics and vertical analysis processing stages present an area for computational improvement. Specifically these analyses are embarrassingly parallel as each oracle is considered independently of the others. While currently designed to run on a single processor these processing stages provide a simple opportunity to divide the workload among multiple processing units. Previous analysis including character encoding and delimiter recognition could also be divided among multiple computational machines, but not as easily as these two stages. Parallelization

provides an obvious opportunity to significantly reduce the runtime required by the brute force, combinatoric, search without any effect on the analysis performed.

6.2.2 Additional Evidence

6.2.2.1 Blank Threshold

The current prototype includes the concept of a minimum threshold. Essentially this property is used to specify the minimum valid count, as a percentage of non-blank records, required for a sequence of characters to be considered a potential field (PF). Another possible piece of evidence worth considering is the blank count. By creating a blank threshold, a maximum number of blank entries as a percentage of the record count, any fields whose blank count is greater than or equal to this threshold could be eliminated or at least marked as uncertain due to a lack of statistical evidence.

In order to test the potential relevance of this form of evidence, updates to the prototype were made in several analysis stages. This was particularly necessary in order to be able to mark fields as uncertain. To include this evidence, changes were made in the combinatoric, conflict resolution, and record update stages. In the combinatoric, or counting, stage the blank counts were tallied and PF that did not comply with their corresponding blank threshold were marked as uncertain. During the conflict resolution stage, these uncertain fields were not even considered as candidates for the record structure. After conflict resolution, the record cleansing stage was invoked per the usual in order to remove inappropriate fields through contextual and cross-reference analysis. Once complete, the record update stage attempts to fill empty, unspecified, regions with available PF including those marked as uncertain, recycling such fields. Once

accomplished the record cleansing stage is invoked again as the final step to analyze any updated fields. These changes to the prototype were necessary to include the concept of uncertainty or ambiguity at a basic level.

The following figures compare the prototype results for three instances: without a blank threshold, eliminating all PF that exceed the blank threshold, and marking PF that exceed the blank threshold as uncertain. Only the results for a few files that showed marked changes are included here. The sample size was 150 records for each file. The actual thresholds were user defined as seemed appropriate based on the minimum threshold, domain definitions, oracle rankings, etc…

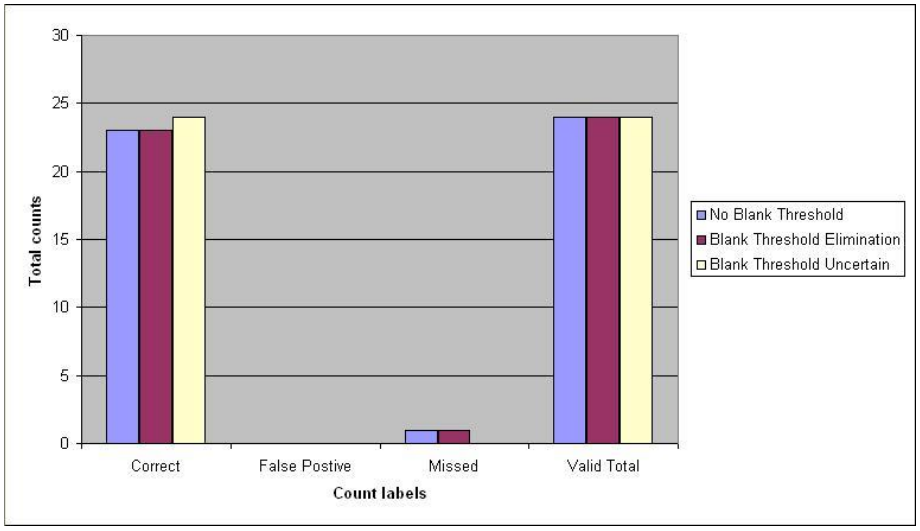

Figure 53: Blank threshold results for File 3.

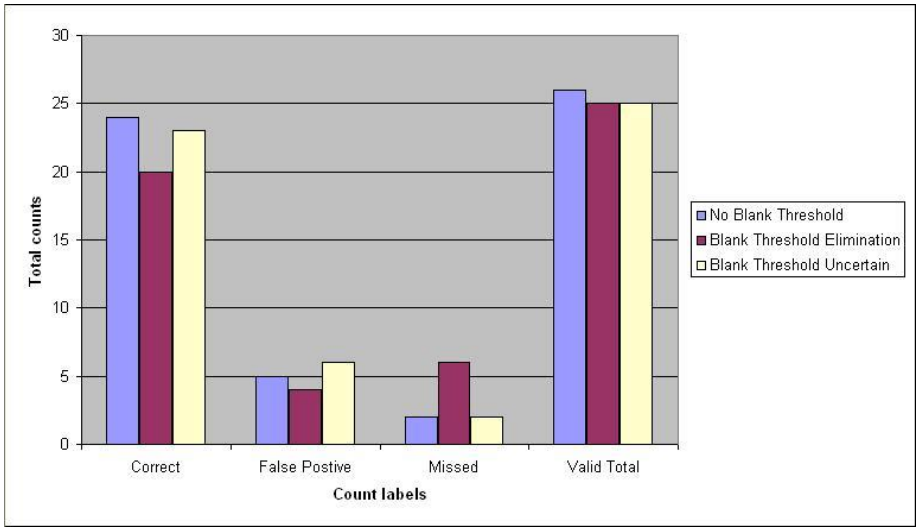

Figure 54: Blank threshold results for File 4.

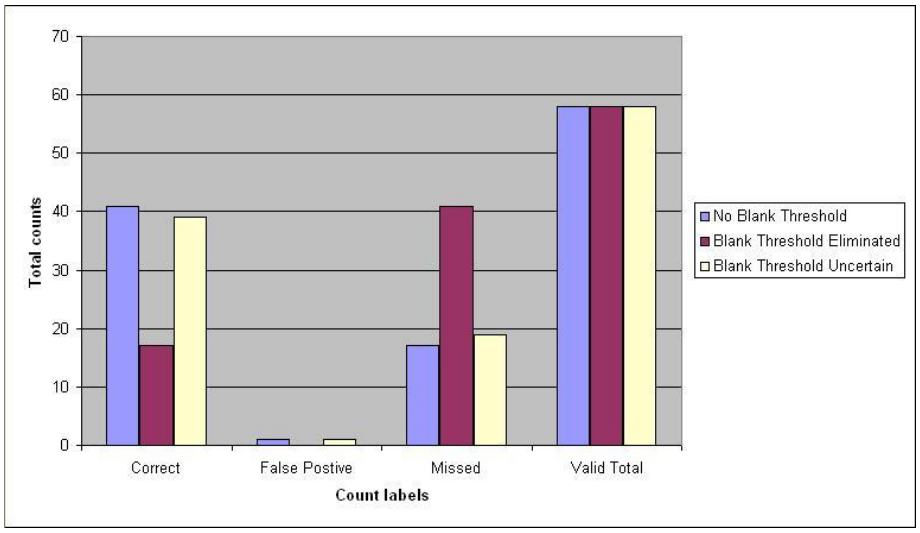

Figure 55: Blank threshold results for File 7.

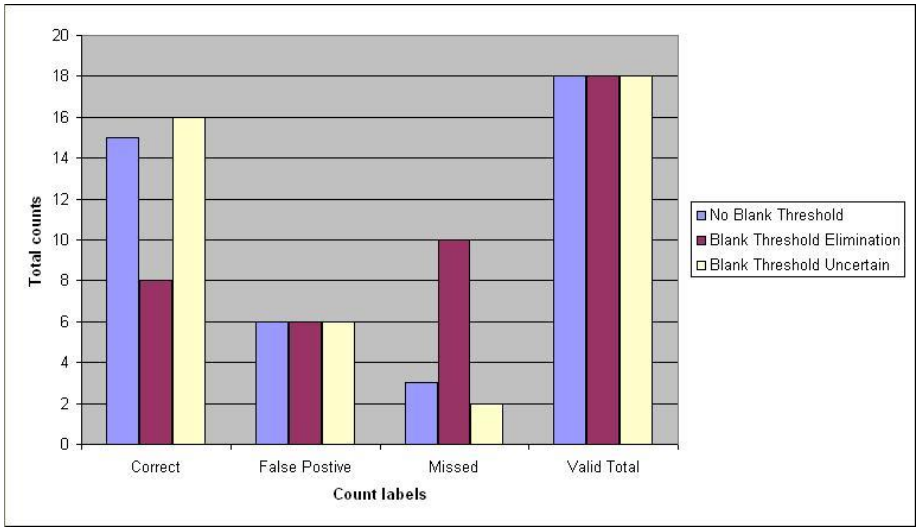

Figure 56: Blank threshold results for File 13.

The usefulness of a blank threshold has only been briefly explored and as a consequence the presented results are introductory and coarse. Current results indicate little to no change. Proper integration of the additional evidence is probably yet to be attained within the layout engine and correct values for each threshold that provide the best statistical results are yet to be determined. These characteristics define the blank threshold to be an area of continued work.

6.2.2.2 Additional Sampling of Sparse Fields

As discussed at various points throughout this paper, PF containing many blank entries are often prone to error. Another possible approach, separate from a blank threshold, is to gather more data from the original data file. This data would involve only the PF under consideration and thus the computationally intensive combinatoric analysis would be unnecessary, making the time footprint of further sampling relatively inconsequential. The desired goal of further sampling would be to extract more non-blank entries from the file lessening the impact of individual entries upon the result.

6.2.2.3 Ternary Oracle Output

Another possible expansion of the prototype involves the oracles' results. As previously defined the oracles return a binary result. The impact if this result were expanded to include three possible values (i.e., definitely not, possibly but not defined by this oracle, and definitely yes) could be significant. Particularly it could be easier to identify incorrect PF based on just a few values. This type of result would have a definite affect on the conflict resolution stage, and possibly on the analysis stages, which would have to be updated to handle this new type of evidence. Also important is the assumption that allows for a certain amount of error in the data, and thus the *definitely not* result must be handled accordingly.

6.2.2.4 Learning from the Data

As many of the oracles are list based, performing validation by comparing an incoming entry against a list of known, valid values, another improvement to the process would be to update the list from entries encountered in the files examined. Ideally, this would be one way to dynamically enhance the prototype's reliability for ever changing content types such as name fields. Other rule based oracle types could also profit from such an addition, specifically those that were adaptive in nature such as Hidden Markov Models or decision trees. Not only could new values be learned, but if the list items or rules were ranked, as suggested in *6.2.1*, then this value could be updated based on frequency counts acquired from the data. Pulling usage statistics directly from the data might prove problematic though as some files might represent a very limited population and the rankings could be skewed. If carefully designed to allow for small population

sizes, the addition of expanding domains suggests the means by which to continually improve the layout engine without making any manual modifications.

6.2.2.5 Feature Integration

Feature collection is a generic term used here to describe the consideration of potentially disparate, relatively high level pieces of evidence. This is a concept used heavily in many of the technologies common to IE and NER. Essentially, this would be necessary when the information associated with a single PF is not enough to make a decision. Instead several properties must be combined to identify the best solution. The layout engine already uses this approach in the final analysis stage (i.e., contextual and cross-reference analysis) where the evidence from potentially multiple fields is compared to remove false positives from the record structure. These forms of analysis define relationships among content types and use the rules of the association as extra evidence from which a better decision can be made. Other eventualities must be considered where these relationship rules do not exist.

A prime example involves data files that are sorted on a single field. In this or similar situations a field may contain a single entry which due to domain overlap could equally well be identified as multiple types: types with no other relational properties from which to make a decision. While vertical analysis may be able to recognize this feature, it would be impossible at that stage to make a determination with respect to the correct course of action. In this circumstance a higher, record level decision must be made combining information in a manner heretofore unconsidered and unattainable in the layout inference approach.

6.2.3 Testing

Another refinement to explore is the sensitivity of individual content types. Current tests indicate the sensitivity of the prototype with respect to the record count, but this can be expanded to determine the number of values generally required to assign a content type to a field. In considering this property the issue of domain overlap must also be allowed for. It is required that not only the correct content type be identified, but also that the evidence be so overwhelming that other similar content types are not identified. This adds to the complexity in defining such an attribute. Tests for a single type would necessarily involve all other content types as an appropriate value must that discriminates the individual type from all others.

Due to restrictions associated with the usage of real data, testing on actual data files was regrettably limited. While this corpus was supplemented with the synthetic data set, its value was limited, a fact already discussed. Expanding the data set to include additional real data is an important next step for this project.

6.2.4 Use Cases

The current approach defines the functionality necessary to providing an automated solution. As such it is ideal for a workflow or batch process, but other useful designs can be envisioned as well. These designs would involve significant alterations to the architecture and possibly to the general approach as well.

6.2.4.1 Interactive Use Case

Between fully manual and fully automated approaches to the layout inference problem lie interactive approaches. As human involvement increases, the level of

automation decreases. The goal of this scenario is to find an equilibrium that optimally combines human intelligence with the computational abilities of a machine [46]. As manual interaction is often an expensive resource, it is important to limit the time users spend waiting on the system. This requires modifications to the architecture. One alteration is to decrease the runtime through adapted heuristics and thresholds and even smaller sample sizes. Since a human is now involved, it is no longer necessary to emphasize high accuracy of the results. It is instead possible to allow for more ambiguity in the intermediary results. The fuzziness in an answer might become evident in field boundaries that are not entirely accurate or multiple content type listings for a single field. Beyond actually improving runtime performance improvements can be simulated by taking advantage of the process' iterative nature. This entails providing intermediary results in stages, performing computations while the user alters and checks the results from the previous stage. Any decrease in accuracy can easily be handled by a user while the layout engine quickly performs identification in a support role.

6.2.4.2 Verification Use Case

Another use case is to use the knowledge contained within the oracles to provide verification for existing layouts. The primary result of the verification process could be statistical measures associated with each field as returned by the oracles and suggested revisions when the returned evidence is incompatible with the existing layout. As the combinatoric stage is reduced or entirely unnecessary, the runtime of this process would be insignificant.

6.3 Summary

In summation, the layout inference problem is the recognition and reporting of certain structural and formatting properties associated with a structured data file. In order to automate recognition, an approach has been described that is statistical in nature. Another key feature to this approach is the use of oracles as expert agents. While building evidence at the various analysis stages throughout the process, these oracles play an important role in recognition. Even though data files are prone to certain features that can make inference difficult (e.g., domain overlap and sparsely populated fields), the implemented prototype has shown the proposed approach to be an appropriate solution. This has been shown through the encouraging results acquired by running the prototype against both synthetic and real data sets. Finally, in recognition of the potential of the proposed approach, several opportunities for improving or expanding the design have been suggested. Ultimately, layout inference provides a challenging problem requiring a knowledge-based, adaptive solution.

REFERENCES

[1] Talburt, J., Chiang, C.-C., Howe, M., Wu, N., Lucero, S., Katkuri, A., Konyaole, C., Nie, M., Sarkar, A., Loghry, J., Nash, D., Stires, J., "A Semiotic Approach to File Layout Inference," *The Acxiom Laboratory for Applied Research (ALAR) Conference on Applied Research in Information Technology*, Conway, Arkansas, USA, February 2008.

[2] Deneke, W., Li, W., Thompson, C., Nash, D., Stires, J., "Towards a Domain Specific Modeling Language for Workflow Specification--Intent Interface and Future Directions," *The Acxiom Laboratory for Applied Research (ALAR) Conference on Applied Research in Information Technology*, Conway, Arkansas, USA, February 2009.

[3] Parsons, T. W., *Introduction to Compiler Construction*, Computer Science Press, New York, 1992.

[4] Abney, S., McAllester, D., Pereira, F. 1999. "Relating Probabilistic Grammars and Automata," In *Proceedings of the 37th Annual Meeting of the Association for Computational Linguistics on Computational Linguistics*, College Park, Maryland, USA, June 1999.

[5] Black, E., Jelinek, F., Lafferty, J., Magerman, D. M., Mercer, R., Roukos, S. 1993, "Towards History-Based Grammars: Using Richer Models for Probabilistic Parsing," In *Proceedings of the 31st annual meeting of the Association for Computational Linguistics on Computational Linguistics*, Columbus, Ohio, USA, June 1993.

[6] Black, E., Lafferty, J., Roukos, S. 1992. "Development and Evaluation of a Broad-Coverage Probabilistic Grammar of English-Language Computer Manuals," In *Proceedings of the 30th annual meeting of the Association for Computational Linguistics on Computational Linguistics*, Newark, Deleware, USA, June 1992.

[7] Albright, A. "Modeling Analogy as Probabilistic Grammar," Presented at *Analogy in Grammar: Form and Acquisition Workshop*, Leipzip, Germany, September 2006.

[8] Crepinsek, M., Mernik, M., Faizan, J., Bryant, B. R., Sprague, A., "Extracting Grammar from Programs: Evolutionary Approach," *ACM SIGPLAN Notices Archive 40(4)*, pp. 39-46.

[9] Hwa, R., "Sample Selection for Statistical Grammar Induction," In *Proceedings of the 2000 Joint SIGDAT Conference on Empirical Methods in Natural Language Processing and Very Large Corpora*, Hong Kong, China, October 2000.

[10] Haghighi, A., Klein, D., "Prototype-Driven Grammar Induction," In *Proceedings of the 21st International Conference on Computational Linguistics and the 44th annual meeting of the Association for Computational Linguistics*, Sydney, Australia, July 2006.

[11] Klein, D., Manning, C. D., "A Generative Constituent-Context Model for Improved Grammar Induction," In *Proceedings of the 40th Annual Meeting on Association for Computational Linguistics*, Philadelphia, Pennsylvania, USA, July 2002.

[12] Mooney, R. J., Bunescu, R., "Mining Knowledge from Text Using Information Extraction," *ACM SIGKDD Explorations Newsletter*, Volume 7 Issue 1, June 2005.

[13] Cowie, J., Lehnert, W., "Information Extraction," *Communications of the ACM*, Volume 39 Issue 1, January 1996.

[14] Borthwick, A., Sterling, J., Agichtein, E., Grishman, R., "Exploiting Diverse Knowledge Sources via Maximum Entropy in Named Entity Recognition," In *Proceedings of the Sixth Workshop on Very Large Corpora*, Montreal, Quebec, August 1998.

[15] Tanabe, L., Wilbur, W., "Tagging Gene and Protein Names in Full Text Articles," In *Proceedings of the ACL-02 workshop on Natural Language Processing in the Biomedical Domain - Volume 3*, Philadelphia, Pennsylvania, USA, July 2002.

[16] Borkar, V., Deshmukh, K., Sarawagi, S., "Automatic Segmentation of Text into Structured Records," *ACM Sigmod*, May 2001.

[17] Cohen, W., Sarawagi, S., "Exploiting Dictionaries in Named Entity Extraction: Combining Semi-Markov Extraction Processes and Data Integration Methods," In *KDD '04: Proceedings of the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, Seattle, Washington, USA, August 2004.

[18] Bikel, D., Miller, S., Schwartz, R., Weischedeld, R., "Nymble: A High-Performance Learning Name Finder," In *Proceedings of the Fifth Conference on Applied Natural Language Processing*, Washington D.C., USA, March 1997.

[19] Russell, S., Norvig, P., *Artificial Intelligence a Modern Approach*, 2nd edition, Upper Saddle River, NJ, Pearson Education, Inc., 2003, p. 549.

[20] Sekine, S., Grishman, R., Shinnou, H., "A Decision Tree Method for Finding and Classifying Names in Japanese Texts," In *Proceedings of the Sixth Workshop on Very Large Corpora*, Montreal, Quebec, Canada, August 1998.

[21] Adelberg, B., "NoDoSE – A Tool for Semi-Automatically Extracting Structured and Semistructured Data from Text Documents," *ACM SIGMOD*, 1998.

[22] Russell, S., Norvig, P., *Artificial Intelligence a Modern Approach*, 2nd edition, Upper Saddle River, NJ, Pearson Education, Inc., 2003, p. 715.

[23] Ibid., pp. 717, 718.

[24] Ibid., p. 101.

[25] Ibid., p. 46.

[26] Hirayama, Y., "A Method for Table Structure Analysis Using DP Matching," In *Proceedings of the 3rd International Conference on Document Analysis and Recognition*, Montreal, Quebec, Canada, August 1995.

[27] Rahgozar, M. and Cooperman, R., "A Graph-Based Table Recognition System," In *Proceedings of Document Recognition III* (IS&T/SPIE electronic imaging'96), San Jose, California, USA, February 1996.

[28] Ramel, J., Crucianu, M., Vincent, N., and Faure, C., "Detection, Extraction and Representation of Tables," In *Proceedings of the 7th International Conference on Document Analysis and Recognition*, Edinburg, Scotland, August 2003.

[29] Shamalian, J., Baird, H., and Wood, T., "A Retargetable Table Reader," In *Proceedings of the 5th International Conference on Document Analysis and Recognition*, Bangalore, India, September 1997.

[30] Embley, E., Hurst, M., Lopresti, D., and Nagy, G., "Table-processing paradigms: a research survey," In *International Journal of Document Analysis*, vol. 8, no. 2, pp. 66-86, 2006.

[31] Tubbs, K., Embley, D., "Recognizing Records from the Extracted Cells of Microfilm Tables," In *Proceedings of the 2002 ACM Symposium on Document Engineering* (DocEng '02), McLean, Virginia, USA, November 2002.

[32] Stavrianou, A., Andritsos, P., Nicoloyannis, N. September 2007. "Overview and Semantic Issues of Text Mining," In *SIGMOG Record 36(3)*. pp. 23-33.

[33] Fan, W., Wallace, L., Rich, S., Zhang, Z. 2006. "Tapping the Power of Text Mining." In *Communications of the ACM 49(9)*, pp. 76-82.

[34] Hearst, M.A. 1999. "Untangling Text Data Mining," In *Proceedings of the 37th Annual Meeting of the Association for Computational Linguistics on Computational Linguistics*, College Park, Maryland, USA, June 1999.

[35] Mozilla.org, "Mozilla Charset Detectors," http://www.mozilla.org/projects/intl/chardet.html.

[36] International Components for Unicode, "Character Set Detection," User Guide for ICU v4.0, http://icu-project.org/userguide/charsetDetection.html.

[37] Russell, S., Norvig, P., *Artificial Intelligence a Modern Approach*, 2nd edition, Upper Saddle River, NJ, Pearson Education, Inc., 2003, p. 111.

[38] Chua, C., Chiang, R., and Lim, E., "Instance-based Attribute Identification in Database Integration," *The VLDB Journal*, 2003.

[39] Allen, J. D., [et all.], The Unicode Consortium, *The Unicode Standard, Version 5.0*, Westford, Massachusetts, 2006, pp. 36-43 (chapter 3).

[40] U.S. Census Bureau, Name Files, 2005, http://www.census.gov/genealogy/names/names_files.html, retrieved November 29, 2007.

[41] U.S. Postal Service, Publication 28 Postal Addressing Standards, 2006, http://pe.usps.gov/text/pub28/welcome.htm, retrieved November 2, 2007.

[42] Resnick, P., The Internet Society, RFC 2822, April 2001, http://tools.ietf.org/html/rfc2822, retrieved November 29, 2007.

[43] Hoag, J., Thompson, C., "A Parallel General-Purpose Synthetic Data Generator," *Sigmod Record*, March 2007.

[44 Canfora, G., Cimitile, A., Garcia, F., Piattini, M., Visaggio, C. A., "Evaluating Advantages of Test Driven Development: a Controlled Experiment with Professionals," In *Proceedings of the 2006 ACM/IEEE International Symposium on Empirical Software Engineering*, Rio de Janeiro, Brazil, September, 2006.

[45] Bhat, T., Nagappan, N., "Evaluating the Efficacy of Test-Driven Development: Industrial Case Studies," In *Proceedings of the 2006 ACM/IEEE International Symposium on Empirical Software Engineering*, Rio de Janeiro, Brazil, September, 2006.

[46] Terveen, L. G., "An Overview of Human-Computer Collaboration," *Knowledge-Based Systems*, 8, pp. 67–81, 1995.

APPENDIX A. CONFIGURATION FILE

Following is an example of a configuration file encoded in XML and used to define the prototype at instantiation. Values associated with each parameter can be viewed as the default values for the prototype.

```
<config>
  <global_parameters>
    <parameter>PrintTextOutput:TRUE</parameter>
    <parameter>RecordsToTest:20</parameter>
    <parameter>AcceptableRecordCount:2</parameter>
    <parameter>OptimalRecordCount:150</parameter>
    <parameter>SamplePartitionCount:5</parameter>
    <parameter>HeaderRecordsToSkip:5</parameter>
    <parameter>SampleSize:100000</parameter>
    <parameter>EbcdicPercentage:0.10</parameter>
    <parameter>LineUpPercentage:0.65</parameter>
    <parameter>HeaderLabelSourceFile:HeaderLabels.dat</parameter>
    <parameter>recordDelimiter:10;13;13,10</parameter>
    <parameter>fieldDelimiter:9;44;124</parameter>
    <parameter>textDelimiter:34;39</parameter>
  </global_parameters>
  <oracles>
    <oracle>
      <qualified_name>oracles.NamePrefixOracle</qualified_name>
      <rank>250</rank>
      <parameters>
        <parameter>TypeName:name prefix</parameter>
        <parameter>MinimumSourceFile:/data/NamePrefix_Limited.dat</parameter>
        <parameter>MaximumSourceFile:/data/NamePrefix_CompleteSorted.dat</parameter>
        <parameter>MaximumLength:25</parameter>
        <parameter>MinimumThreshold:0.68</parameter>
        <parameter>MaximumBlankPercentage:0.98</parameter>
        <parameter>MaximumNumberPercentage:0.95</parameter>
        <parameter>Grouping:1</parameter>
      </parameters>
    </oracle>
    <oracle>
      <qualified_name>oracles.FirstNameOracle</qualified_name>
      <rank>1000</rank>
      <parameters>
        <parameter>TypeName:first name</parameter>
        <parameter>MinimumSourceFile:/data/FN_Top2000.dat</parameter>
        <parameter>MaximumSourceFile:/data/FN_13785.dat</parameter>
        <parameter>MaximumLength:50</parameter>
        <parameter>MinimumThreshold:0.60</parameter>
        <parameter>MaximumBlankPercentage:0.15</parameter>
        <parameter>Grouping:1</parameter>
      </parameters>
    </oracle>
    <oracle>
      <qualified_name>oracles.MiddleNameOracle</qualified_name>
      <rank>500</rank>
      <parameters>
        <parameter>TypeName:middle name</parameter>
        <parameter>MaximumLength:0</parameter>
        <parameter>MinimumThreshold:0.90</parameter>
        <parameter>MaximumBlankPercentage:1.00</parameter>
        <parameter>Grouping:1</parameter>
      </parameters>
    </oracle>
    <oracle>
      <qualified_name>oracles.LastNameOracle</qualified_name>
```

```
  <rank>1000</rank>
  <parameters>
    <parameter>TypeName:last name</parameter>
    <parameter>MinimumSourceFile:/data/LN_Top2000.dat</parameter>
    <parameter>MaximumSourceFile:/data/LN_88698.dat</parameter>
    <parameter>MaximumLength:50</parameter>
    <parameter>MinimumThreshold:0.60</parameter>
    <parameter>MaximumBlankPercentage:0.15</parameter>
    <parameter>Grouping:10</parameter>
  </parameters>
</oracle>
<oracle>
  <qualified_name>oracles.NameSuffixOracle</qualified_name>
  <rank>250</rank>
  <parameters>
    <parameter>TypeName:name suffix</parameter>
    <parameter>MinimumSourceFile:/data/NameSuffix_Limited.dat</parameter>
    <parameter>MaximumSourceFile:/data/NameSuffix_CompleteSorted.dat</parameter>
    <parameter>MaximumLength:25</parameter>
    <parameter>MinimumThreshold:0.70</parameter>
    <parameter>MaximumBlankPercentage:0.98</parameter>
    <parameter>MaximumNumberPercentage:0.95</parameter>
    <parameter>Grouping:10</parameter>
  </parameters>
</oracle>
<oracle>
  <qualified_name>oracles.FullNameOracle</qualified_name>
  <rank>1000</rank>
  <parameters>
    <parameter>TypeName:full name</parameter>
    <parameter>MaximumLength:60</parameter>
    <parameter>MinimumThreshold:0.60</parameter>
    <parameter>MaximumBlankPercentage:0.15</parameter>
    <parameter>Grouping:11</parameter>
  </parameters>
</oracle>
<oracle>
  <qualified_name>oracles.StreetNumberOracle</qualified_name>
  <rank>30</rank>
  <parameters>
    <parameter>TypeName:street number</parameter>
    <parameter>MaximumLength:15</parameter>
    <parameter>MinimumThreshold:0.50</parameter>
    <parameter>MaximumBlankPercentage:0.25</parameter>
    <parameter>Grouping:100</parameter>
  </parameters>
</oracle>
<oracle>
  <qualified_name>oracles.DirectionalOracle</qualified_name>
  <rank>50</rank>
  <parameters>
    <parameter>TypeName:directional</parameter>
    <parameter>MaximumLength:20</parameter>
    <parameter>MinimumThreshold:0.60</parameter>
    <parameter>MaximumBlankPercentage:1.00</parameter>
    <parameter>Grouping:100</parameter>
  </parameters>
</oracle>
<oracle>
  <qualified_name>oracles.StreetNameOracle</qualified_name>
  <rank>750</rank>
  <parameters>
    <parameter>TypeName:street name</parameter>
    <parameter>MinimumSourceFile:/data/StreetNames_Top800.dat</parameter>
    <parameter>MaximumSourceFile:/data/StreetNames_Top50000.dat</parameter>
    <parameter>MaximumLength:50</parameter>
    <parameter>MinimumThreshold:0.65</parameter>
    <parameter>MaximumBlankPercentage:0.50</parameter>
    <parameter>Grouping:100</parameter>
  </parameters>
</oracle>
```

```
<oracle>
  <qualified_name>oracles.StreetSuffixOracle</qualified_name>
  <rank>750</rank>
  <parameters>
    <parameter>TypeName:street suffix</parameter>
    <parameter>MaximumSourceFile:/data/StreetSuffixes_CompleteSorted.dat</parameter>
    <parameter>MaximumLength:35</parameter>
    <parameter>MinimumThreshold:0.60</parameter>
    <parameter>MaximumBlankPercentage:0.80</parameter>
    <parameter>Grouping:100</parameter>
  </parameters>
</oracle>
<oracle>
  <qualified_name>oracles.AddressLineOneOracle</qualified_name>
  <rank>1000</rank>
  <parameters>
    <parameter>TypeName:address line one</parameter>
    <parameter>MaximumLength:60</parameter>
    <parameter>MinimumThreshold:0.60</parameter>
    <parameter>MaximumBlankPercentage:0.15</parameter>
    <parameter>Grouping:100</parameter>
  </parameters>
</oracle>
<oracle>
  <qualified_name>oracles.UnitDesignatorOracle</qualified_name>
  <rank>500</rank>
  <parameters>
    <parameter>TypeName:unit designator</parameter>
    <parameter>MaximumSourceFile:/data/UnitDesignator_CompleteSorted.dat</parameter>
    <parameter>MaximumLength:35</parameter>
    <parameter>MinimumThreshold:0.60</parameter>
    <parameter>MaximumBlankPercentage:1.00</parameter>
    <parameter>Grouping:100</parameter>
  </parameters>
</oracle>
<oracle>
  <qualified_name>oracles.SecondaryRangeNumberOracle</qualified_name>
  <rank>20</rank>
  <parameters>
    <parameter>TypeName:unit number</parameter>
    <parameter>MaximumLength:15</parameter>
    <parameter>MinimumThreshold:0.10</parameter>
    <parameter>MaximumBlankPercentage:1.00</parameter>
    <parameter>Grouping:100</parameter>
  </parameters>
</oracle>
<oracle>
  <qualified_name>oracles.AddressLineTwoOracle</qualified_name>
  <rank>800</rank>
  <parameters>
    <parameter>TypeName:address line two</parameter>
    <parameter>MaximumLength:50</parameter>
    <parameter>MinimumThreshold:0.15</parameter>
    <parameter>MaximumBlankPercentage:1.00</parameter>
    <parameter>Grouping:100</parameter>
  </parameters>
</oracle>
<oracle>
  <qualified_name>oracles.AddressLineOracle</qualified_name>
  <rank>1000</rank>
  <parameters>
    <parameter>TypeName:address line</parameter>
    <parameter>MaximumLength:75</parameter>
    <parameter>MinimumThreshold:0.60</parameter>
    <parameter>MaximumBlankPercentage:0.15</parameter>
    <parameter>Grouping:100</parameter>
  </parameters>
</oracle>
<oracle>
  <qualified_name>oracles.CityNameOracle</qualified_name>
  <rank>1000</rank>
```

```
  <parameters>
    <parameter>TypeName:city</parameter>
    <parameter>MinimumSourceFile:/data/Cities_Top7500.dat</parameter>
    <parameter>MaximumSourceFile:/data/Cities_CompleteSorted.dat</parameter>
    <parameter>MaximumLength:50</parameter>
    <parameter>MinimumThreshold:0.60</parameter>
    <parameter>MaximumBlankPercentage:0.10</parameter>
    <parameter>Grouping:1000</parameter>
  </parameters>
</oracle>
<oracle>
  <qualified_name>oracles.StateOracle</qualified_name>
  <rank>800</rank>
  <parameters>
    <parameter>TypeName:state</parameter>
    <parameter>MaximumSourceFile:/data/States_CompleteSorted.dat</parameter>
    <parameter>MaximumLength:50</parameter>
    <parameter>MinimumThreshold:0.60</parameter>
    <parameter>MaximumBlankPercentage:0.10</parameter>
    <parameter>Grouping:1000</parameter>
  </parameters>
</oracle>
<oracle>
  <qualified_name>oracles.ZipcodeOracle</qualified_name>
  <rank>500</rank>
  <parameters>
    <parameter>TypeName:zipcode</parameter>
 <parameter>CrossReferenceSourceFile:/data/StateCityZip_SortedByState.dat</parameter>
    <parameter>MaximumSourceFile:/data/ZipCodes_CompleteSorted.dat</parameter>
    <parameter>MaximumLength:15</parameter>
    <parameter>MinimumThreshold:0.60</parameter>
    <parameter>CrossReferenceThreshold:0.70</parameter>
    <parameter>MaximumBlankPercentage:0.20</parameter>
    <parameter>Grouping:1000</parameter>
  </parameters>
</oracle>
<oracle>
  <qualified_name>oracles.PhoneNumberOracle</qualified_name>
  <rank>250</rank>
  <parameters>
    <parameter>TypeName:phone number</parameter>
    <parameter>MaximumSourceFile:/data/NpaNxxStateZip.dat</parameter>
    <parameter>MaximumLength:20</parameter>
    <parameter>MinimumThreshold:0.50</parameter>
    <parameter>MaximumBlankPercentage:0.98</parameter>
    <parameter>Grouping:10000</parameter>
  </parameters>
</oracle>
<oracle>
  <qualified_name>oracles.EmailOracle</qualified_name>
  <rank>800</rank>
  <parameters>
    <parameter>TypeName:email</parameter>
    <parameter>MaximumSourceFile:/data/EMAIL-US-TLD.dat</parameter>
    <parameter>MaximumLength:100</parameter>
    <parameter>MinimumThreshold:0.15</parameter>
    <parameter>MaximumBlankPercentage:1.00</parameter>
    <parameter>Grouping:10000</parameter>
  </parameters>
</oracle>
<oracle>
  <qualified_name>oracles.BooleanOracle</qualified_name>
  <rank>75</rank>
  <parameters>
    <parameter>TypeName:boolean</parameter>
    <parameter>MaximumLength:1</parameter>
    <parameter>MinimumThreshold:0.95</parameter>
    <parameter>MaximumBlankPercentage:0.10</parameter>
    <parameter>Grouping:10000</parameter>
  </parameters>
</oracle>
```

```
  </oracles>
</config>
```

APPENDIX B. OUTPUT

Included in this appendix are example outputs for a fully delimited and a hybrid file (the output for a fully fixed file is essentially the same as that of a hybrid file). As mentioned the end user should determine the correct format for the output. Currently the prototype includes functionality for a formatted text output and XML. While the meaning of most of the headings is evident some of the abbreviations contained in the formatted text output are not necessarily intuitive.

- J is the justification of the associated field. Can assume values 'L' for left, 'R' for right, and 'C' for center justified.
- SP is the start position of the associated field: character offset for both fixed file types and relative field position for fully delimited files.
- EP is the end position of the associated field: character offset for both fixed file types and not applicable for fully delimited files.
- LEN is the length, EP minus SP plus one, of the associated field: character count for both fixed file types and not applicable for fully delimited files.
- PC is the primary count determined during the field identification stage.
- SC is the secondary count determined after any applicable oracle's domains have been reduced to address the domain overlap problem. The value is negative if the oracle's domain is not reduced.
- BC is the blank count. This is how many fields contain only whitespace.

Given the entry:

```
1.)   full name              L    1   27  27 --   150  141    0
```

…the first field in the record structure has been associated with the “full name” content type. The field is left justified, begins at offset 1, ends at position 27, has length 27, has a primary count of 150, a secondary count of 141, and there were no blank entries.

B.1 Hybrid File

The following is the text output of the prototype for a hybrid file.

```
FILE NAME: Hybrid.txt
RUN TIME: 13922
CHARACTER ENCODING: ASCII
FILE TYPE: 2 - hybrid
HEADER: false
RECORD DELIMITER
  POSSIBLE: 13,10;13;10
  FOUND: 13,10
FIELD DELIMITER
  POSSIBLE: 9;44;124
  FOUND:
TEXT DELIMITER ( string literals )
  POSSIBLE: 34;39
  FOUND:
RECORDS EXAMINED: 150
RECORD LENGTH: 107
LAYOUT ---
       CONTENT TYPE NAME         J   SP   EP LEN       PC   SC   BC
   1.)   full name               L    1   27  27 --   150  141    0
   2.)   address line one        L   28   54  27 --   128   85    0
   3.)   address line two        L   55   81  27 --    80   -1   70
   4.)   city                    L   82   99  18 --   150  100    0
   5.)   state                   L  100  102   3 --   150   -1    0
   6.)   zipcode                 L  103  107   5 --   150   -1    0
----------

POTENTIAL FIELD LOCATIONS ---
          CONTENT TYPE NAME         J   SP   EP LEN       PC   SC   BC
  4-  1.)   last name               C    8   27  20 --    93   77    0
  4-  2.)   last name               L   33   37   5 --   149   66    0
  6-  1.)   full name               L    1   27  27 --   150  141    0
  7-  1.)   street number           R   16   29  14 --   118   -1    0
  7-  2.)   street number           L   28   31   4 --   150   -1    0
  7-  3.)   street number           L   31   32   2 --   137   -1   13
  7-  4.)   street number           L   60   74  15 --    80   -1   70
  7-  5.)   street number           L  103  107   5 --   150   -1    0
  8-  1.)   directional             L   33   37   5 --   149   -1    0
  9-  1.)   street name             L   82   99  18 --   119   26    0
 10-  1.)   street suffix           L   56   57   2 --    80   -1   70
 11-  1.)   address line one        L   28   54  27 --   128   85    0
 13-  1.)   unit number             L    1    4   4 --   150   -1    0
 13-  2.)   unit number             L    5    5   1 --   141   -1    9
 13-  3.)   unit number             L    6    6   1 --   121   -1   29
 13-  4.)   unit number             L    7    7   1 --   107   -1   43
 13-  5.)   unit number             L    8    8   1 --   111   -1   39
 13-  6.)   unit number             L    9    9   1 --   130   -1   20
 13-  7.)   unit number             L   10   10   1 --   140   -1   10
 13-  8.)   unit number             L   11   11   1 --   146   -1    4
 13-  9.)   unit number             L   12   12   1 --   124   -1   26
 13- 10.)   unit number             L   13   13   1 --   110   -1   40
 13- 11.)   unit number             L   14   14   1 --    84   -1   66
 13- 12.)   unit number             R   16   29  14 --   118   -1    0
 13- 13.)   unit number             L   28   31   4 --   149   -1    0
```

```
 13- 14.)   unit number            L   28   32   5 --   150   -1    0
 13- 15.)   unit number            L   33   35   3 --   150   -1    0
 13- 16.)   unit number            L   36   38   3 --   149   -1    1
 13- 17.)   unit number            L   39   40   2 --   150   -1    0
 13- 18.)   unit number            L   41   41   1 --   143   -1    7
 13- 19.)   unit number            L   42   42   1 --   138   -1   12
 13- 20.)   unit number            L   43   43   1 --   122   -1   28
 13- 21.)   unit number            L   44   44   1 --   118   -1   32
 13- 22.)   unit number            L   45   45   1 --   124   -1   26
 13- 23.)   unit number            L   46   46   1 --   126   -1   24
 13- 24.)   unit number            L   47   47   1 --   138   -1   12
 13- 25.)   unit number            L   48   48   1 --   122   -1   28
 13- 26.)   unit number            L   49   49   1 --   118   -1   32
 13- 27.)   unit number            L   50   50   1 --   100   -1   50
 13- 28.)   unit number            L   51   51   1 --    74   -1   76
 13- 29.)   unit number            L   52   52   1 --    60   -1   90
 13- 30.)   unit number            C   54   55   2 --    76   -1   56
 13- 31.)   unit number            L   55   57   3 --    80   -1   70
 13- 32.)   unit number            L   60   74  15 --    80   -1   70
 13- 33.)   unit number            L   82   83   2 --   150   -1    0
 13- 34.)   unit number            L   84   84   1 --   147   -1    3
 13- 35.)   unit number            L   85   85   1 --   141   -1    9
 13- 36.)   unit number            L   86   86   1 --   137   -1   13
 13- 37.)   unit number            L   87   87   1 --   131   -1   19
 13- 38.)   unit number            L   88   88   1 --   107   -1   43
 13- 39.)   unit number            L   89   89   1 --    83   -1   67
 13- 40.)   unit number            R   98   99   2 --   147   -1    0
 13- 41.)   unit number            L  100  107   8 --   150   -1    0
 14-  1.)   address line two       L   55   81  27 --    80   -1   70
 15-  1.)   address line           L   28   54  27 --   128   85    0
 16-  1.)   city                   L   33   37   5 --   113   66    0
 16-  2.)   city                   L   82   99  18 --   150  100    0
 17-  1.)   state                  L   19   20   2 --     1   -1  149
 17-  2.)   state                  R   98  100   3 --   147   -1    0
 17-  3.)   state                  L  100  102   3 --   150   -1    0
 18-  1.)   zipcode                L  103  107   5 --   150   -1    0
 21-  1.)   boolean                L   18   18   1 --     4   -1  146
-----------------------------

CONTENT TYPES:
name prefix
first name
middle name
last name
name suffix
full name
street number
directional
street name
street suffix
address line one
unit designator
unit number
address line two
address line
city
state
zipcode
phone number
email
boolean
```

The following is the XML output of the prototype for a hybrid file.

```
<?xml version="1.0" encoding="UTF-8"?>
<layout>
  <filename>Hybrid.txt</filename>
  <runtime>13547</runtime>
```

```
<encoding>ASCII</encoding>
<filetype>2</filetype>
<header>FALSE</header>
<delimiters>
  <record>
    <possible>13,10;13;10</possible>
    <found>13,10</found>
  </record>
  <field>
    <possible>9;44;124</possible>
    <found></found>
  </field>
  <textquote>
    <possible>34;39</possible>
    <found></found>
  </textquote>
</delimiters>
<recordsexamined>
  <count>150</count>
</recordsexamined>
<contenttypes>
  <type>name prefix</type>
  <type>first name</type>
  <type>middle name</type>
  <type>last name</type>
  <type>name suffix</type>
  <type>full name</type>
  <type>street number</type>
  <type>directional</type>
  <type>street name</type>
  <type>street suffix</type>
  <type>address line one</type>
  <type>unit designator</type>
  <type>unit number</type>
  <type>address line two</type>
  <type>address line</type>
  <type>city</type>
  <type>state</type>
  <type>zipcode</type>
  <type>phone number</type>
  <type>email</type>
  <type>boolean</type>
</contenttypes>
<record>
  <length>107</length>
  <fields>
    <field>
      <position>
        <start>1</start>
        <end>27</end>
        <length>27</length>
      </position>
      <headerlabel>
        <label></label>
        <probability>0.00</probability>
      </headerlabel>
      <contenttype>full name</contenttype>
      <justification>LEFT</justification>
      <fillcharacter></fillcharacter>
      <validcount>
        <primary>150</primary>
        <secondary>141</secondary>
        <blankcount>0</blankcount>
      </validcount>
      <certain>FALSE</certain>
    </field>
    <field>
      <position>
        <start>28</start>
        <end>54</end>
        <length>27</length>
```

```
  </position>
  <headerlabel>
    <label></label>
    <probability>0.00</probability>
  </headerlabel>
  <contenttype>address line one</contenttype>
  <justification>LEFT</justification>
  <fillcharacter></fillcharacter>
  <validcount>
    <primary>128</primary>
    <secondary>85</secondary>
    <blankcount>0</blankcount>
  </validcount>
  <certain>FALSE</certain>
</field>
<field>
  <position>
    <start>55</start>
    <end>81</end>
    <length>27</length>
  </position>
  <headerlabel>
    <label></label>
    <probability>0.00</probability>
  </headerlabel>
  <contenttype>address line two</contenttype>
  <justification>LEFT</justification>
  <fillcharacter></fillcharacter>
  <validcount>
    <primary>80</primary>
    <secondary></secondary>
    <blankcount>70</blankcount>
  </validcount>
  <certain>FALSE</certain>
</field>
<field>
  <position>
    <start>82</start>
    <end>99</end>
    <length>18</length>
  </position>
  <headerlabel>
    <label></label>
    <probability>0.00</probability>
  </headerlabel>
  <contenttype>city</contenttype>
  <justification>LEFT</justification>
  <fillcharacter></fillcharacter>
  <validcount>
    <primary>150</primary>
    <secondary>100</secondary>
    <blankcount>0</blankcount>
  </validcount>
  <certain>FALSE</certain>
</field>
<field>
  <position>
    <start>100</start>
    <end>102</end>
    <length>3</length>
  </position>
  <headerlabel>
    <label></label>
    <probability>0.00</probability>
  </headerlabel>
  <contenttype>state</contenttype>
  <justification>LEFT</justification>
  <fillcharacter></fillcharacter>
  <validcount>
    <primary>150</primary>
    <secondary></secondary>
```

```
          <blankcount>0</blankcount>
        </validcount>
        <certain>FALSE</certain>
      </field>
      <field>
        <position>
          <start>103</start>
          <end>107</end>
          <length>5</length>
        </position>
        <headerlabel>
          <label></label>
          <probability>0.00</probability>
        </headerlabel>
        <contenttype>zipcode</contenttype>
        <justification>LEFT</justification>
        <fillcharacter></fillcharacter>
        <validcount>
          <primary>150</primary>
          <secondary></secondary>
          <blankcount>0</blankcount>
        </validcount>
        <certain>FALSE</certain>
      </field>
    </fields>
  </record>
  <potentialfieldlocations>
    …
  </potentialfieldlocations>
</layout>
```

B.2 Fully Delimited File

The following is the text output of the prototype for a fully delimited file.

```
FILE NAME: Delimited.txt
RUN TIME: 266
CHARACTER ENCODING: ASCII
FILE TYPE: 3 - fully delimited
HEADER: true
RECORD DELIMITER
  POSSIBLE: 13,10;13;10
  FOUND: 13,10
FIELD DELIMITER
  POSSIBLE: 9;44;124
  FOUND: 44
TEXT DELIMITER ( string literals )
  POSSIBLE: 34;39
  FOUND:
RECORDS EXAMINED: 118
RECORD LENGTH: 12
LAYOUT ---
       CONTENT TYPE NAME           SP       PC   SC   BC
   1.)   * UNSPECIFIED *            1 --     0   -1    0
   2.)   name prefix                2 --    43   -1   75
   3.)   first name                 3 --   118  113    0
   4.)   middle name                4 --     0   -1    0
   5.)   last name                  5 --   118  113    0
   6.)   name suffix                6 --    15   -1  103
   7.)   address line one           7 --   112   75    0
   8.)   city                       8 --   118   81    0
   9.)   state                      9 --   118   -1    0
  10.)   zipcode                   10 --   118   -1    0
  11.)   * UNSPECIFIED *           11 --     0   -1    0
```

```
  12.)   * UNSPECIFIED *           12 --     0   -1    0
----------

POTENTIAL FIELD LOCATIONS ---
          CONTENT TYPE NAME           SP       PC   SC   BC
  1-  1.)   name prefix                2 --    43   -1   75
  2-  1.)   first name                 3 --   118  113    0
  2-  2.)   first name                 5 --   101   63    0
  4-  1.)   last name                  3 --   113   58    0
  4-  2.)   last name                  5 --   118  113    0
  5-  1.)   name suffix                6 --    15   -1  103
  7-  1.)   street number              1 --   118   -1    0
  7-  2.)   street number             10 --   118   -1    0
  7-  3.)   street number             11 --   118   -1    0
  9-  1.)   street name                3 --   118   45    0
  9-  2.)   street name                5 --   114  101    0
  9-  3.)   street name                8 --    94   26    0
 11-  1.)   address line one           7 --   112   75    0
 13-  1.)   unit number                1 --   118   -1    0
 13-  2.)   unit number                4 --   118   -1    0
 13-  3.)   unit number                6 --     8   -1  103
 13-  4.)   unit number               10 --   118   -1    0
 13-  5.)   unit number               11 --   118   -1    0
 13-  6.)   unit number               12 --   118   -1    0
 15-  1.)   address line               7 --   112   75    0
 16-  1.)   city                       3 --    99   60    0
 16-  2.)   city                       5 --   103   87    0
 16-  3.)   city                       8 --   118   81    0
 17-  1.)   state                      9 --   118   -1    0
 18-  1.)   zipcode                   10 --   118   -1    0
-----------------------------

CONTENT TYPES:
name prefix
first name
middle name
last name
name suffix
full name
street number
directional
street name
street suffix
address line one
unit designator
unit number
address line two
address line
city
state
zipcode
phone number
email
boolean
```

The following is the XML output of the prototype for a fully delimited file.

```
<?xml version="1.0" encoding="UTF-8"?>
<layout>
  <filename>Delimited.txt</filename>
  <runtime>516</runtime>
  <encoding>ASCII</encoding>
  <filetype>3</filetype>
  <header>TRUE</header>
  <delimiters>
    <record>
      <possible>13,10;13;10</possible>
      <found>13,10</found>
```

```
  </record>
  <field>
    <possible>9;44;124</possible>
    <found>44</found>
  </field>
  <textquote>
    <possible>34;39</possible>
    <found></found>
  </textquote>
</delimiters>
<recordsexamined>
  <count>118</count>
</recordsexamined>
<contenttypes>
  <type>name prefix</type>
  <type>first name</type>
  <type>middle name</type>
  <type>last name</type>
  <type>name suffix</type>
  <type>full name</type>
  <type>street number</type>
  <type>directional</type>
  <type>street name</type>
  <type>street suffix</type>
  <type>address line one</type>
  <type>unit designator</type>
  <type>unit number</type>
  <type>address line two</type>
  <type>address line</type>
  <type>city</type>
  <type>state</type>
  <type>zipcode</type>
  <type>phone number</type>
  <type>email</type>
  <type>boolean</type>
</contenttypes>
<record>
  <length>12</length>
  <fields>
    <field>
      <position>
        <start>1</start>
        <end></end>
        <length></length>
      </position>
      <headerlabel>
        <label>Seq#</label>
        <probability>1.00</probability>
      </headerlabel>
      <contenttype>* UNSPECIFIED *</contenttype>
      <justification>LEFT</justification>
      <fillcharacter></fillcharacter>
      <validcount>
        <primary>0</primary>
        <secondary></secondary>
        <blankcount>0</blankcount>
      </validcount>
      <certain>FALSE</certain>
    </field>
    <field>
      <position>
        <start>2</start>
        <end></end>
        <length></length>
      </position>
      <headerlabel>
        <label>NPrefix</label>
        <probability>0.00</probability>
      </headerlabel>
      <contenttype>name prefix</contenttype>
      <justification>LEFT</justification>
```

```
  <fillcharacter></fillcharacter>
  <validcount>
    <primary>43</primary>
    <secondary></secondary>
    <blankcount>75</blankcount>
  </validcount>
  <certain>FALSE</certain>
</field>
<field>
  <position>
    <start>3</start>
    <end></end>
    <length></length>
  </position>
  <headerlabel>
    <label>FName</label>
    <probability>0.15</probability>
  </headerlabel>
  <contenttype>first name</contenttype>
  <justification>LEFT</justification>
  <fillcharacter></fillcharacter>
  <validcount>
    <primary>118</primary>
    <secondary>113</secondary>
    <blankcount>0</blankcount>
  </validcount>
  <certain>FALSE</certain>
</field>
<field>
  <position>
    <start>4</start>
    <end></end>
    <length></length>
  </position>
  <headerlabel>
    <label>MI</label>
    <probability>0.00</probability>
  </headerlabel>
  <contenttype>middle name</contenttype>
  <justification>LEFT</justification>
  <fillcharacter></fillcharacter>
  <validcount>
    <primary>0</primary>
    <secondary></secondary>
    <blankcount>0</blankcount>
  </validcount>
  <certain>FALSE</certain>
</field>
<field>
  <position>
    <start>5</start>
    <end></end>
    <length></length>
  </position>
  <headerlabel>
    <label>LName</label>
    <probability>0.00</probability>
  </headerlabel>
  <contenttype>last name</contenttype>
  <justification>LEFT</justification>
  <fillcharacter></fillcharacter>
  <validcount>
    <primary>118</primary>
    <secondary>113</secondary>
    <blankcount>0</blankcount>
  </validcount>
  <certain>FALSE</certain>
</field>
<field>
  <position>
    <start>6</start>
```

```
      <end></end>
      <length></length>
    </position>
    <headerlabel>
      <label>NSuffix</label>
      <probability>0.00</probability>
    </headerlabel>
    <contenttype>name suffix</contenttype>
    <justification>LEFT</justification>
    <fillcharacter></fillcharacter>
    <validcount>
      <primary>15</primary>
      <secondary></secondary>
      <blankcount>103</blankcount>
    </validcount>
    <certain>FALSE</certain>
  </field>
  <field>
    <position>
      <start>7</start>
      <end></end>
      <length></length>
    </position>
    <headerlabel>
      <label>Address1</label>
      <probability>0.00</probability>
    </headerlabel>
    <contenttype>address line one</contenttype>
    <justification>LEFT</justification>
    <fillcharacter></fillcharacter>
    <validcount>
      <primary>112</primary>
      <secondary>75</secondary>
      <blankcount>0</blankcount>
    </validcount>
    <certain>FALSE</certain>
  </field>
  <field>
    <position>
      <start>8</start>
      <end></end>
      <length></length>
    </position>
    <headerlabel>
      <label>City</label>
      <probability>0.00</probability>
    </headerlabel>
    <contenttype>city</contenttype>
    <justification>LEFT</justification>
    <fillcharacter></fillcharacter>
    <validcount>
      <primary>118</primary>
      <secondary>81</secondary>
      <blankcount>0</blankcount>
    </validcount>
    <certain>FALSE</certain>
  </field>
  <field>
    <position>
      <start>9</start>
      <end></end>
      <length></length>
    </position>
    <headerlabel>
      <label>ST</label>
      <probability>0.00</probability>
    </headerlabel>
    <contenttype>state</contenttype>
    <justification>LEFT</justification>
    <fillcharacter></fillcharacter>
    <validcount>
```

```
        <primary>118</primary>
        <secondary></secondary>
        <blankcount>0</blankcount>
      </validcount>
      <certain>FALSE</certain>
    </field>
    <field>
      <position>
        <start>10</start>
        <end></end>
        <length></length>
      </position>
      <headerlabel>
        <label>Zipcode</label>
        <probability>0.00</probability>
      </headerlabel>
      <contenttype>zipcode</contenttype>
      <justification>LEFT</justification>
      <fillcharacter></fillcharacter>
      <validcount>
        <primary>118</primary>
        <secondary></secondary>
        <blankcount>0</blankcount>
      </validcount>
      <certain>FALSE</certain>
    </field>
    <field>
      <position>
        <start>11</start>
        <end></end>
        <length></length>
      </position>
      <headerlabel>
        <label>Salary</label>
        <probability>1.00</probability>
      </headerlabel>
      <contenttype>* UNSPECIFIED *</contenttype>
      <justification>LEFT</justification>
      <fillcharacter></fillcharacter>
      <validcount>
        <primary>0</primary>
        <secondary></secondary>
        <blankcount>0</blankcount>
      </validcount>
      <certain>FALSE</certain>
    </field>
    <field>
      <position>
        <start>12</start>
        <end></end>
        <length></length>
      </position>
      <headerlabel>
        <label>Date</label>
        <probability>1.00</probability>
      </headerlabel>
      <contenttype>* UNSPECIFIED *</contenttype>
      <justification>LEFT</justification>
      <fillcharacter></fillcharacter>
      <validcount>
        <primary>0</primary>
        <secondary></secondary>
        <blankcount>0</blankcount>
      </validcount>
      <certain>FALSE</certain>
    </field>
  </fields>
</record>
<potentialfieldlocations>
  …
</potentialfieldlocations></layout>
```