SANDIA REPORT

SAND2010-1756
Unlimited Release
Printed April 2010

SAPLE: Sandia Advanced Personnel Locator Engine

Michael J. Procopio

Prepared by
Sandia National Laboratories
Albuquerque, New Mexico 87185 and Livermore, California 94550

Sandia is a multiprogram laboratory operated by Sandia Corporation, a Lockheed Martin Company, for the United States Department of Energy's National Nuclear Security Administration under Contract DE-AC04-94-AL85000.

Approved for public release; further dissemination unlimited.

Sandia National Laboratories

Issued by Sandia National Laboratories, operated for the United States Department of Energy by Sandia Corporation.

NOTICE: This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government, nor any agency thereof, nor any of their employees, nor any of their contractors, subcontractors, or their employees, make any warranty, express or implied, or assume any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represent that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government, any agency thereof, or any of their contractors or subcontractors. The views and opinions expressed herein do not necessarily state or reflect those of the United States Government, any agency thereof, or any of their contractors.

Printed in the United States of America. This report has been reproduced directly from the best available copy.

Available to DOE and DOE contractors from
U.S. Department of Energy
Office of Scientific and Technical Information
P.O. Box 62
Oak Ridge, TN 37831

Telephone: (865) 576-8401
Facsimile: (865) 576-5728
E-Mail: reports@adonis.osti.gov
Online ordering: <http://www.osti.gov/bridge>

Available to the public from
U.S. Department of Commerce
National Technical Information Service
5285 Port Royal Rd
Springfield, VA 22161

Telephone: (800) 553-6847
Facsimile: (703) 605-6900
E-Mail: orders@ntis.fedworld.gov
Online ordering: <http://www.ntis.gov/help/ordermethods.asp?loc=7-4-0#online>

SAPLE: Sandia Advanced Personnel Locator Engine

Michael J. Procopio
Sandia National Laboratories
Albuquerque, NM 87185
Email: mprocopio@gmail.com

Abstract

We present the Sandia Advanced Personnel Locator Engine (SAPLE) web application, a directory search application for use by Sandia National Laboratories personnel. SAPLE's purpose is to return Sandia personnel "results" as a function of user search queries, with its mission to make it easier and faster to find people at Sandia. To accomplish this, SAPLE breaks from more traditional directory application approaches by aiming to return the correct set of results while placing minimal constraints on the user's query. Two key features form the core of SAPLE: advanced search query interpretation and inexact string matching. SAPLE's query interpretation permits the user to perform compound queries when typing into a single search field; where able, SAPLE *infers* the type of field that the user intends to search on based on the value of the search term. SAPLE's inexact string matching feature yields a high-quality ranking of personnel search results even when there are no exact matches to the user's query. This paper explores these two key features, describing in detail the architecture and operation of SAPLE. Finally, an extensive analysis on logged search query data taken from an 11-week sample period is presented.

Acknowledgment

Much of the software development for the production version of SAPLE was done by Jesse Flemming, whose software engineering effort helped bring SAPLE to reality. Bill Cook provided vision and support for SAPLE which directly resulted in its deployment at the laboratory. Sam Cancilla and Tracy Walker were instrumental in coordinating the initial SAPLE deployment as well as ongoing enhancements. Brian Byers created the current user interface for SAPLE, with input from both SWIFT team usability personnel as well as Alisa Bandlow. Joe Lewis supported SAPLE at important points in its deployment, especially during enterprise/ADP review. Cara Corey and Matt Schrager facilitated SAPLE's deployment, and Cara also presented SAPLE at the Department of Energy InterLab 2009 conference.

Ken Osburn's load testing yielded important insights. The Middleware team, especially Mathew Anderson, supported SAPLE on their Web Logic servers from the earliest prototypes to its current lab-wide realization. The Enterprise Database Administration team, especially Peggy Schroeder, Mike Mink, and Bill Mertens, provided crucial support for SAPLE's database components. Andrew Steele led SAMS, the first project to use SAPLE's external search web service. David Gallegos provided helpful managerial support and perspective. Philip Kegelmeyer, Travis Bauer, Bryan Kennedy, Brian Jones, Pete Oelschlaeger, and Andy Tzou all provided suggestions and guidance along the way.

To all of these individuals, and to the other unnamed "Friends of SAPLE" who supported the SAPLE concept as it evolved over the years: thank you.

Contents

1	Introduction	7
1.1	Emergence of Advanced Web Applications	7
1.2	Related Work	8
1.3	Searching with SAPLE	10
1.4	Organization of the Paper	10
2	Architecture and Implementation.....	13
2.1	Personnel Index	14
2.2	Organization Index	15
2.3	Motivation for Indexing	15
2.4	Creation of the Indexes	16
2.5	Querying the Index: An Example.....	16
2.6	Implementation Technologies	17
3	Search Query Interpretation.....	19
3.1	Inference of Query Terms	21
3.2	Cautious Query Inference	22
3.3	Implementation of Query Parsing.....	23
4	Scoring and Ranking Search Results.....	25
4.1	SAPLE Fuzzy Search Algorithms	25
4.2	Combination of Algorithms	27
4.3	Examples of Algorithm Ensemble Combination	27
5	SAPLE Web Service	29
5.1	Web Service Interface and Specification.....	29
5.2	Example Client Application: Software Asset Management System (SAMS).....	30
6	Analytics and Usage Analysis	31
6.1	General Usage and Load	31
6.2	Search Query Type Breakdown.....	32
6.3	Search Usage by User	36
6.4	User Type Breakdown	36
6.5	SAPLE Search Sources	38
6.6	SAPLE Computational Performance	39
7	Conclusions and Future Work	43
	References	45

Figures

1	The SAPLE Home Page (http://saple.sandia.gov).	10
2	Screenshots from the SAPLE application.	11
3	Screenshots for Facebook.com (a), Google.com (b), and Amazon.com (c) websites demonstrating use of approximate string matching. In all three cases, inexact (or fuzzy) string matching algorithms are used to determine more likely user queries, and the results for the predicted user query are instead returned. Instead of returning “zero results found,” the correct set of results for the user’s intended query is displayed.	12
4	SAPLE three-tiered application architecture, containing a presentation layer, an application layer, and a data layer.	13
5	Legacy Directory application architecture. In contrast to a three-tiered application architecture (Figure 4), here, the layers are less distinct; business rules may be contained in the presentation and/or data layers.	14
6	Multiple search fields (input text boxes) in the Legacy Directory personnel search application.	19
7	Google.com interpretation of multiple query parameters from a single text string input	20
8	SAPLE’s Advanced Search screen. The values entered in by the user for each text box are consolidated into a single string with multiple <code>field:value</code> pairs, illustrated in Figure 7. ...	20

9	Example of a compound query (i.e., query with multiple field:value pairs)	21
10	Example of a “search space collision,” in which a ambiguous value (not specifically associated with a field name), if the field name is <i>inferred</i> , could be associated with multiple fields. In this case, the user query 6710 could be in reference to both Building 6710, or Organization 6710, both of which are real entities.	22
11	Example XML result set from the SAPLE web service (left). XML Schema Excerpt (saple.xsd) (right).	30
12	Screenshot of SAMS Application. In this scenario, the query org 10501 lab staff has been passed to the SAPLE web service via AJAX (JavaScript), and the XML result set returned by SAPLE has been parsed and presented to the user in a dropdown list.	30
13	SAPLE Usage for Monday, 9/21/2009 – Friday, 12/4/2009.	31
14	SAPLE Usage for Thursday, 12/3/2009.	32
15	Breakdown of SAPLE Searches by type. Type names correspond to entries in Table 5. The data show that single-token name searches and organization-related searches form the majority of the queries processed by SAPLE.	33
16	SAPLE search usage (frequency) for the top 5,000 SAPLE users	36
17	SAPLE Searches by Title (Grouped)	37
18	Breakdown of SAPLE Searches by Title	37
19	SAPLE Algorithm Computation Time by Search Type	40
20	Distribution of SAPLE Computation Times, Fastest 99%. Approximately 1% of the slowest queries have been removed as outliers. The distribution exhibits clear trimodality, which is correlated to the actual type of search performed and resulting computational complexity. . .	41

Tables

1	Example Index Entries	16
2	Example Alias Table for Job Title (plural entries removed for brevity)	23
3	Score computation table for user query “Tayler” vs. candidate matches “Taylor” and “Tyler”	28
4	Score computation table for user query “Consillo” vs. candidate matches “Cancilla” and “Castillo”	28
5	SAPLE Searches by Type	32
6	Top 50 SAPLE Search Queries ^a	34
7	Random 200 SAPLE Search Queries	35
8	SAPLE Searches by Title (All Titles)	37
9	SAPLE Search Sources	38
10	SAPLE Computation times and count by query type.	40

1 Introduction

SAPLE (Sandia Advanced Personnel Locator Engine) is a web-based personnel search engine, also referred to as a directory application. Its purpose is to return Sandia personnel “results” as a function of user search queries. A variety of personnel-related query mechanisms are supported; for example, “Chris Smith” would find and return all such individuals at Sandia, while “5527” would return a roster of results of all personnel in Sandia’s Organization 05527. As of this writing, SAPLE is deployed laboratory-wide at Sandia and services an approximate query load of 10,000 personnel searches daily.

SAPLE’s mission is to make it easier and faster to find people at Sandia. To accomplish this, SAPLE breaks from more traditional directory application approaches by aiming to return the correct set of results while placing very minimal constraints on a user’s query. This mission, which is focused on the user experience, largely informs the requirements of SAPLE and drives many engineering and architectural decisions.

SAPLE’s aim to make finding people at Sandia easier and faster results in some key differences versus traditional directory approaches. Most notably, legacy approaches enforce certain constraints on the user’s query (and hence the user experience). For example, search queries must be in `lastname, firstname` format; the user must not omit a comma; spelling must be exact; search terms such as phone number or job title are not supported; compound queries are not supported; and so on.

Beyond concerns about the user experience, the primary drawbacks to legacy directory applications generally pertain to search quality concerns, i.e., retrieving correct results for the user:

- It may require the user several attempts to identify the person for whom they’re searching;
- It may require a significant amount of the user’s time to identify the correct person from a large, unranked list of results;
- Given what vague information a user knows about some individual in question, it may not be possible to find that person with traditional directory searches.

Existing legacy personnel directory approaches at Sandia have been generally effective in providing a nominal search capability, which is certainly improved in many ways over manual hardcopy directory-based methods. SAPLE, as shown in Figure 1, represents a natural, next iteration for this capability, in much the same way as other information technology services at Sandia have also improved over time (for example, Techweb, Sandia’s Intranet). In general, SAPLE is designed less like a legacy web tool, and more like a contemporary search engine such as Google. In particular, SAPLE uses advanced algorithms operating on an *index*, an internally constructed representation of raw personnel data, from which a *ranked list* of all applicable results (personnel) is computed and returned.

1.1 Emergence of Advanced Web Applications

In recent years, a number of factors have come together to motivate and enable the design and deployment of a search application like SAPLE.

- There is now a mature, generally robust infrastructure for developing and supporting contemporary web applications like SAPLE at Sandia. Corporate-maintained Middleware services such as Web Logic Server and Enterprise Database services such as SQL Server are central components of this infrastructure. Moreover, Sandia employs key personnel to maintain these resources, which allow application developers to spend more time focusing on their primary task, software engineering.
- High-level, mature, and widely adopted programming languages, such as the Java programming language, are now common and supported in corporate environments. These languages have improved

over the course of 15 years, to include very important and enabling programming constructs, such as those required for multithreading (concurrent) programming. These languages (Java in particular) are well supported by the corporate server environments available at Sandia.

- SAPLE and other search engines require the use of advanced search algorithms to generate their internal indexes, search results, and associated ranking of output. These algorithms, many of which appear only in the recent literature from the past 15 years, enable increasingly sophisticated and relevant output from search engines.
- Sandia has placed great focus on hiring and retaining Computer Scientists, who are comfortable operating in environments which require knowledge in many areas. Beyond up-to-date software engineering using contemporary programming languages (such as Java) and recent best practices (unit testing, agile development methodologies), these individuals have experience with algorithm design, analysis, and optimization, and in many cases, experience with large-scale applications in high performance computing contexts.
- Increasingly, users are demanding richer and richer capabilities and user experiences.

1.2 Related Work

The use of computational methods for improving directory-based lookup services dates back to the 1980s [12], while methods that help link related last names together for census purposes have been described in the early 1900s [7]. More sophisticated solutions have been developed over time and for varying purposes [14]. The *record linkage* problem [11] and the related *duplicate record detection* problem [4] are well studied, and pose challenges to domains as diverse as informational retrieval and databases [20] to biology [3]. The *approximate string matching* function in SAPLE is a composite of many algorithms found in the above references. Excellent surveys of direct relevance to SAPLE’s approximate string matching task are provided [18], [1], and [10].

String-Similarity Metrics

The basic principles of fuzzy string matching, also known as *inexact* or *approximate* string matching, have been known for some time (here, string refers to a series of one or more letters). For example, string-based similarity metrics are used in inexact text searching command-line utilities built into the Unix and Linux operating systems, most notably, **agrep** (for *approximate grep* [17]).

As another prevalent example, the spell checker feature found on many modern day word processors is based primarily on the “string distance” metric, in which words “closest to” some unrecognized word are identified and presented to the user as suggested spellings for the misspelled word. This spell check capability dates back to research from the late 1970’s; implementations of spell checkers first appeared on personal desktop computers in the early 1980s, becoming common later that decade.

Traditionally, spelling suggestions were determined by taking those words in a known dictionary with small *Levenshtein distance* values to the misspelled word. For example, “keybord” and “keyboard” have a very small Levenshtein distance (only one letter needs to be added to “keybord” in order to match “keyboard”), so “keyboard” would be suggested by the spell checker. Meanwhile, words in the dictionary with large Levenshtein distance to the misspelled word would not be included in the list of suggestions.

The Levenshtein distance is a canonical string-distance algorithm and is more commonly known as “edit distance.” There are many variants of the basic edit distance algorithm; for example, different penalties can be applied for different types of transformations (insertions, deletions, and swaps) when trying to make a misspelled word match some known word. Other string-based similarity metrics are known; for example, *longest common subsequence* and *longest common substring* both return values that are indicative of string

similarity. More esoteric approaches are also known, for example *Smith-Waterman* [3], *Jaro-Winkler* [16], and *n*-gram [15] metrics. For a survey of these and related string-based similarity metrics, see [2].

Phonetic Algorithms

In addition to the string-based similarity metrics described above, so-called *phonetic* methods are also known. These methods consider not just the letters in the string, but also how the string is pronounced. Such phonetic-based approaches are used for many tasks today, for example, contemporary spell checking modules that utilize both string-based and phonetic-based similarity measures [8].

One of the earliest examples of a rule-based phonetic method is the Soundex algorithm [9], a crude method patented in the early 1900s and used by the United States Census during that time [7]. With this method, “procopio” and “persopio” both return an encoding of p621, and would be declared a phonetic match. A number of issues occur in practical use of Soundex leading to low-quality phonetic matching, and Soundex is therefore not in widespread use today.

More contemporary phonetic approaches include the Double Metaphone algorithm by Lawrence Philips [13]; the operation is similar to Soundex, but uses a much larger encoding rule set and is generally considered more effective. Other methods, such as Phonix [5] [6] and the New York State Identification and Intelligence System (NYSIIS) phonetic code [14] are also known. An important hybrid algorithm that combines both distance-based and phonetic-based approaches is the Editex algorithm [19]. This algorithm is an extension of the Levenshtein (basic edit distance) algorithm, but phonetic similarity is considered when computing the cost of each transformation (substitution, swap, etc.) and the final similarity score (string distance) is thus a combination of both the transformations required and the letters involved.

Phonetic algorithms can be very powerful but suffer from being generally *language specific*. This is due to the various phonetic encodings and equivalences in such algorithms being tuned to the pronunciation of one particular language. Because sounds vary from language to language, it is very difficult to create a universal phonetic algorithm, and that is why many of the algorithms noted above have direct equivalents in different languages. Consider the double consonant sequence “ll.” In Spanish-language names, this sequence is almost always pronounced like the English “y.” In English, and even in other closely related Latin-derived languages such as Italian, it is pronounced as the consonant “l.” Many other examples exist.

Certainly, it is possible to create or even modify an existing algorithm with this phonetic equivalence. However, if not done carefully, such an algorithm will do better for some names, but will negatively impact the search quality on others. Nonetheless, as shown in Table 6, five out of the six most frequent name queries in SAPLE’s search logs are for Spanish-language last names. This finding supports investing future effort on improving the phonetic-based search algorithms used in SAPLE to perform better for Spanish-language last names.

Examples of Approximate String Matching

Although somewhat rare in the early 2000s, examples of approximate string matching had become widespread by late that same decade. Most of the major websites which accept typed user queries as input use some form of approximate string matching. For example, as shown in Figure 3, the websites Facebook.com, Google.com, and Amazon.com all have this capability built in. In each of these three examples, poor results would have been returned for the user’s initial query on an “exact match” basis. Therefore, a query close to the initial query (in terms of edit distance or phonetic similarity) that returned good search results would instead be suggested; in these cases, the suggested query is actually used instead of the user’s initial query. This same behavior—returning high-quality search results for the user’s *intended* query—is found in these and many other major web sites found on the Internet (including eBay, Yahoo!, YouTube, Wikipedia, and so on).

1.3 Searching with SAPLE

Figure 1 shows the home page for the SAPLE application, located at <http://saple.sandia.gov>. Figure 2 shows two screenshots of SAPLE in action. In the first, the search results for user query `gensalis tech staff ipoc` are given in Picture View. In the second, a roster listing for Organization 10501 is shown in List View, as returned by the user query `10501`.

Figure 1: The SAPLE Home Page (<http://saple.sandia.gov>).

1.4 Organization of the Paper

This paper is divided into two general sections; the first half introduces SAPLE, while the second half presents an analysis on the logged query data for an 11-week sample period. In Section 2, SAPLE’s three-tiered application architecture is presented in detail, as well as the structure of its internal *indexes*. In Section 3, SAPLE’s methods for interpreting the user’s query are examined. Section 4 presents SAPLE’s internal search and ranking algorithms. SAPLE’s web service, which permits external application developers to use SAPLE, is described in Section 5. An extensive analysis on logged query data is presented in Section 6. Finally, conclusions and areas for future work are given in Section 7.

Download as [Text](#) | [CSV \(Excel\)](#)

saple [Advanced Search](#) **Display Options:** ☒ Picture ☒ List ☐ Include Visitors

Searched for **gensalis** jobtitle:tech staff building:ipoc Results 1 - 3 of 3

Berlanda C. Gonzales [Details](#)

Email Office (505) 844-4149 Location

STAFF-TECH Fax (505) 284-2380 User ID

[Org 9548](#) | [Reporting and Planning Systems](#) [MySite Profile](#) | [9548 Manager](#)

Dublin Gonzales [Details](#)

Email Office (505) 284-5860 Location

STAFF-TECH Fax (505) 844-7240 User ID

[Org 9543](#) | [SW Engineering](#) [MySite Profile](#) | [9543 Manager](#)

Mark A. Gonzales [Details](#)

Email Office (505) 844-4799 Location

STAFF-TECH Fax (505) 844-7240 Mobile User ID

[Org 9543](#) | [SW Engineering](#) [MySite Profile](#) | [9543 Manager](#)

(a) SAPLE Screenshot for user query `gensalis tech staff ipoc`. Here, the user intends to show all technical staff (STAFF-TECH) in the IPOC building with “Gonzales” as their last name. The search is still successful, despite misspelling Gonzales as “gensalis.”

Download as [Text](#) | [CSV \(Excel\)](#)

saple [Advanced Search](#) **Display Options:** ☐ Picture ☒ List ☐ Include Visitors

Searched for org:10501 Results 1 - 14 of 14

10501 Accounting Services [Edit Classification Data](#)

MANAGER	Office	Mobile	Fax						
SEC/OAA	Office	Mobile	Fax						
Name	Email	Org	Office	Fax	Mailstop	Bldg/Rm	SNL ID	Type	Title
Anderson, Carol I.		10501	(505) 844-3350	(505) 844-9845	1382				
Bauer, Donna J.		10501	(505) 844-6956	(505) 844-9845	1382				
Brooks, Traci		10501	(505) 845-7534	(505) 844-9845	1382				
Christ, Heather J.		10501	(505) 845-8350	(505) 844-9845	1382				
Cline, Tim E.		10501	(505) 845-7917	(505) 844-9845	1382				
Ferguson, Jr., Waylon B.		10501	(505) 844-3057	(505) 844-9845	1382				
Galbraith, Kathleen		10501	(505) 284-0926	(505) 844-9845	1382				
Gonzales, Jennifer		10501	(505) 845-1008	(505) 844-9845	1382				
Gutierrez, Gertrude E.		10501	(505) 844-2764	(505) 844-9845	1382				
Martinez, Rita D.		10501	(505) 284-2356	(505) 844-9845	1382				
Owens, Polly E.		10501	(505) 844-3559	(505) 844-9845	1382				
Tyler, Pamela J.		10501	(505) 844-9523	(505) 844-9845	1382				
Wareham, Amanda		10501	(505) 844-0167	(505) 844-9845	1382				
Woolley, Amy S.		10501	(505) 284-9442	(505) 844-9845	1382				

(b) SAPLE Screenshot in “List View” showing the org roster listing for Organization 10501, returned by a user query of 10501.

Figure 2: Screenshots from the SAPLE application.

(a) Facebook.com screenshot showing approximate string matching; here, “sharron percopio” is correctly mapped to “sharon procopio.”

(b) Google.com screenshot showing approximate string matching; here, “james camrin” is correctly mapped to “james cameron.”

(c) Amazon.com screenshot showing approximate string matching. Here, the user query “nonparametric statistical methods” is correctly mapped to “nonparametric statistical methods.”

Figure 3: Screenshots for Facebook.com (a), Google.com (b), and Amazon.com (c) websites demonstrating use of approximate string matching. In all three cases, inexact (or fuzzy) string matching algorithms are used to determine more likely user queries, and the results for the predicted user query are instead returned. Instead of returning “zero results found,” the correct set of results for the user’s intended query is displayed.

2 Architecture and Implementation

SAPLE is based on a standard, three-tier web application architecture comprising a *data* layer, an *application* or *business logic* layer, and a *presentation* layer (Figure 4). The presentation layer is the part of the application that the user interacts with through their browser; in this layer, queries are submitted by the user, and results are displayed in response. The query is interpreted according to rules found in the application/logic layer; query interpretation is discussed in Section 3. Also in the application/logic layer, various search algorithms, described in Section 4, are used to compute the similarity of the user's query to each person in the *personnel index*. This personnel index (described below) is created from underlying personnel data by querying Sandia's personnel databases using stored procedures, all of which is done in the data layer. Finally, the scores are tabulated, sorted, and the ranked list of results is returned to the presentation layer for display to the user.

This approach differs from the simpler architecture used by legacy directory applications. These simpler approaches—for example, Sandia's legacy directory application—do not contain a true application layer, as illustrated in Figure 5. Instead, in the simpler legacy approach, user queries are formulaically translated into SQL query *where clauses*, and passed directly to the data layer as a database query. The raw results from the query (rows in the database selected by the where clause) are then returned to the user (in this case, with minimal processing).

Figure 4: SAPLE three-tiered application architecture, containing a presentation layer, an application layer, and a data layer.

Figure 5: Legacy Directory application architecture. In contrast to a three-tiered application architecture (Figure 4), here, the layers are less distinct; business rules may be contained in the presentation and/or data layers.

2.1 Personnel Index

The inclusion of an application layer enables most of the advanced functionality present in SAPLE. In particular, it is in this layer that the *personnel index* is created. This index is a condensed, optimized, in-memory representation of the information for all personnel searchable by SAPLE. Each entry in the index corresponds to one individual, and comprises standard attributes about them (location, phone number, etc.) as well as custom values computed from the standard attributes when the index is created.

In the application layer, the index is created by a call to a stored procedure exposed in the Microsoft SQL Server SAPLE database maintained by Sandia's Enterprise Database Administration team. The stored procedure contains a simple SQL SELECT statement, which for efficiency returns a subset of *selected* attributes (i.e., database columns or fields) for *selected* personnel (i.e., database rows or entries). The stored procedure retrieves records from an SQL Server *view* of the ePerson table. This view enables access to an exported subset of records and attributes retrieved from a larger, more comprehensive set of personnel records (namely, PeopleSoft).

The raw ePerson table (view) is quite large, containing 209,549 entries for the production ePerson export at last count. As noted above, for efficiency, the stored procedure only returns a subset of this information. In particular, the attributes chosen to return in the stored procedure's results include only those required by the SAPLE application itself (these appear in the SELECT statement).

Importantly, only a fraction of the actual entries in the ePerson table are returned. Currently, 21,706 entries of the 209,549 are returned (10%). This is because only *active* personnel (to include those personnel on leaves of absence) are included in SAPLE results; former employees including retirees are not included.

The actual SELECT statement used in SAPLE's stored procedure for retrieving personnel records is shown below:

```

SELECT SNL_ID, SL_OPERID, SL_FIRST_NAME, SL_MIDDLE_NAME, SL_LAST_NAME, SL_NAME_SUFFIX,
SL_NAME, SL_NICKNAME, SL_JOBTITLE_ABBRV, SL_MAILSTOP, SL_FAX, SL_CELL, SL_AREA_MSTR_CD,
SL_SITE_MSTR_CD, SL_STRUCT_MSTR_CD, SL_ROOM_MSTR_CD, SL_WORK_PHONE, SL_DEPTNAME_ABBRV,
SL_JOBTITLE, SL_JOB_FAMILY, SL_JOBFAMDES, SL_PGRNM, SL_URL, SL_LOCATION_ID,
SL_REL_STATUS, SL_REL_NAME, SL_REL_SNAME, SL_EMPL_TYPE, SL_EMAIL_ADDR

FROM extcopy.dbo.eperson

WHERE ( (SL_REL_STATUS = 'A' OR SL_REL_STATUS = 'L'))

ORDER BY SL_NAME

```

2.2 Organization Index

SAPLE includes in its results certain organizational information, such as department name, homepage web URL, manager of record, and secretary of record. Because this information is not available in ePerson, a separate, smaller index is created, similar to the personnel index. This *organization index* is generated along with and in the same manner as the personnel index. It currently contains 1,085 rows.

This information is generated by a stored procedure that creates an SQL join of the SL_DEPT_URL_VW table with the SL_ORGN table using a field common to both, the organization number. Table SL_DEPT_URL_VW includes name and homepage web URL attributes, while table SL_ORGN contains the SNL IDs of the managers and secretaries of record for each organization.

When generating results, entries in the organization index are referenced as needed, using the organization number as the lookup key. For example, when a user searches on “5527,” the organization name, web address, manager, and secretary are retrieved from the organization index and displayed accordingly, followed by the staff in that organization (whose information in turn comes directly from the personnel index).

The SELECT statement used in the stored procedure to generate the organization index is shown below:

```

SELECT extcopy.dbo.sl_orgn.SL_ORGNUM, extcopy.dbo.SL_DEPT_URL_VW.DESCR,
extcopy.dbo.SL_DEPT_URL_VW.SL_URL, extcopy.dbo.SL_DEPT_URL_VW.SL_DEPT_DESCR,
extcopy.dbo.sl_orgn.MANAGER_SNL_ID, extcopy.dbo.sl_orgn.SL_SECRETARY_ID

FROM extcopy.dbo.SL_DEPT_URL_VW INNER JOIN extcopy.dbo.sl_orgn ON
extcopy.dbo.SL_DEPT_URL_VW.DESCRSHORT = extcopy.dbo.sl_orgn.SL_ORGNUM

WHERE extcopy.dbo.sl_orgn.SL_STATUS_CD = 'A'

```

2.3 Motivation for Indexing

The indexes are maintained in main memory (RAM) on the web application server, enabling the search algorithms to operate much faster than if the index were kept on disk. In fact, performance is the primary motivating factor of creating and maintaining a custom search index in RAM. In particular, the using an index prevents multiple repeated calls to the backend databases; not only is making such a call for each user query inefficient (resulting in increased latency), but such calls could become burdensome to the database servers handling the requests. With an index, only a single call to the databases is needed when the index is created or refreshed (which may happen several times per day).

Maximizing precomputation is a key component motivating creating and maintaining an internal index. The index permits as much as possible to be precomputed when the index is created (e.g., phonetic encodings of names), improving latency (response time) for use queries. This in turn enables certain search algorithms to operate very efficiently; because these algorithms can be computationally intensive, it is important that as much as possible be computed ahead of time during index creation.

For example, SAPLE uses phonetic (sound-based) search algorithms that can identify that “humfrey” sounds like “humphrey.” To accomplish this, the *phonetic encodings* are computed; in this fictitious example, both could be represented as “HMFR.” Rather than re-compute the same phonetic encoding for each name in the index every time a search is done, this only needs to be done once, at index creation time. The only cost is a small amount of extra space, but this reduces search time (latency) for the end user vastly (e.g., by an order of magnitude). Not all algorithms are able to be optimized in this manner, but many are.

Additional details regarding SAPLE’s search algorithms are presented in Section 4.

2.4 Creation of the Indexes

Creation of the personnel and organizational indexes happens in two ways. First, the indexes are created whenever the application is freshly deployed (i.e., when no index currently exists). The same behavior effectively happens when the corporate servers hosting SAPLE reboot, and the application effectively gets re-deployed.

SAPLE is also designed to refresh its index periodically throughout the day. The frequency of rebuilding the indexes can be easily controlled in the program source code. Currently, the indexes are set to refresh every four hours, with the intention being to pick up “live” changes to personnel data throughout the day. The frequency of the underlying ePerson data refresh is only once per day at present. Nonetheless, refreshing every four hours ensures that SAPLE “picks up” the nightly-refreshed ePerson data within a reasonable time window.

Actual creation of the two indexes takes approximately 15 seconds. Therefore, when SAPLE determines the indexes are over four hours old, it triggers a rebuild process in the background using Java-based multi-threading/concurrent programming constructs. This monitoring check itself happens only upon a user query. While the new indexes are being refreshed in the background, current searches continue to be handled using the existing index. When the background index build process is complete, the old indexes are instantly replaced by the new ones, transparently to users, and SAPLE queries after this point will use the updated indexes.

2.5 Querying the Index: An Example

Table 1 shows a figurative excerpt from an index for four Sandia staff members. The first two columns, **first_name** and **last_name**, show actual attribute data (columns) from the underlying database tables. The last two columns show the phonetic representations of both the first and last name for each person, as computed by the Double Metaphone algorithm (see Section 4.1). Computation of these is time intensive, and only needs to be done once, ahead of time, during index creation.

Now, consider a user’s hypothetical query of **Mickeal Percoppia**. Before comparison, the phonetic encoding for each term in this query is computed—MKL PRKP—and then compared to the precomputed phonetic encodings of each the name in the index. Because the phonetic encodings computed for the user’s query match those computed earlier in the index, the “Michael Procopio” index entry would be scored highly in this example.

Table 1: Example Index Entries

first_name	last_name	org	first_phonetic	last_phonetic
Bettina	Schimanski	01234	PTN	XMNSK
Samuel	Cancilla	01234	SML	KNSL
Michael	Procopio	01234	MKL	PRKP
Jesse	Flemming	01234	JS	FLMNK

2.6 Implementation Technologies

In addition to the three layers in the SAPLE architecture, Figure 4 also shows the associated technologies for each layer. In the presentation layer, a user uses a web browser, such as Google Chrome, Mozilla FireFox, or Microsoft Internet Explorer to view the main SAPLE search page. This is the user interface. The web page is programmed in HTML (HyperText Markup Language), which is created by a user interface designer. The designer also creates accompanying CSS, or cascading style sheets, which controls how the page appears in terms of layout, fonts, spacing, colors, and so on. The application developers add in JavaScript code, which enables certain rich user interactions in the presentation layer, for example, switching between views and handling other user input. Finally, all of this is rendered on the *server side* by Java Server Pages, or JSPs, which get compiled to Java *servlets* on the back end. These JSP pages are responsible for returning structured blocks of HTML code back to the user, and will vary depending on the user’s query and other parameters. For example, the SAPLE JSP page will return HTML to render a “picture view” (with badge photos) or a tabular “list view,” depending on the user preference. Browser *cookies* are used to store user preferences such as this in between browser *sessions*.

The application layer is written entirely in Java Enterprise, formerly known as J2EE. This is essentially the core SAPLE application, and it is hosted on a Java Enterprise *application server*. At Sandia, corporate resources provide and support the use of Oracle Corporation’s *Web Logic Server*. The particular server—which at Sandia, is divided among development (*webdev*), quality (*webqual*), and production (*webprod*) *server instances*, controls the execution of server side Java code associated with SAPLE. This includes the SAPLE Java code to create the internal personnel and organization indexes, the code needed for search query interpretation (see Section 3), as well as the search algorithms themselves.

Finally, in the data layer, Java modules known as Java Database Connectivity, or JDBC, components are called to execute *stored procedures* hosted on the *database server*. Here, Microsoft SQL Server is used, as this is a corporate-supported solution at Sandia; other database options, such as Oracle, are also available. The stored procedures are written in Microsoft’s *transact-SQL* extended database query language. The stored procedures query the underlying data source very efficiently. Here, this data source is a *view* of selected personnel data for selected individuals, that is built nightly from the core Sandia personnel data repository, PeopleSoft. The stored procedures return a set of results, which are processed by the JDBC components, and returned to the application layer where they are transformed accordingly to form an index.

3 Search Query Interpretation

SAPLE supports searching on a variety of *fields*; for example, searches can be performed by specifying not just name, but also organization, mailstop, building, employee type, etc. Moreover, *compound* searches can be performed, in which only personnel who match *all* specified query terms are returned (for example, “10221 lab staff”).

In a traditional personnel search approach, multiple text boxes, one for each field, are presented to the user; the user must identify which fields and textboxes to place the various terms relevant to their search. This is shown below in Figure 6.

The screenshot shows a web-based search interface for the Legacy Directory. At the top, a red warning message states: "This directory is to be used for official Sandia National Laboratories business only. Its use by unauthorized persons is prohibited." Below this, a link for "Click field links for [HELP](#)" is provided. The main section contains a list of search criteria, each with a corresponding text input box: "Search by [Name](#)", "Search by [Nickname](#)", "Search by [Organization](#)", "Search by [Building](#)", "Search by [Phone](#)", "Search by [Fax](#)", "Search by [Mail Stop](#)", "Search by [Email ID](#)", "Search by [Title](#)", and "Search by [Type](#)". The "Type" field includes a dropdown menu with a downward arrow. Below the input boxes, there are three checkboxes: "Employees" (checked), "Contractors" (checked), and "Others" (unchecked). At the bottom left, there are two buttons: "submit" and "clear".

Figure 6: Multiple search fields (input text boxes) in the Legacy Directory personnel search application.

In contrast, search engines such as Google use the *single search box* approach; Figure 7 provides an example. Here, the actual search query text is `procopio filetype:pdf site:sandia.gov`; in fact, the real search query is “procopio,” but additional search specifiers—in this case, file type and website—can also be provided. These `field:value` pairs are automatically identified and parsed from the query, and influence the operation of the search.

Like Google, SAPLE adopts the single search box approach described above. In this sense, SAPLE’s single search box approach is not novel, nor is its use of key/value pairs. SAPLE’s use of a single search box was a decision made in support of SAPLE’s aim to make searching easier for the user, and has been generally well received. Nonetheless, some users may prefer the multiple-field method for searching, perhaps in part as it provides “prompts” for fields a user could search on. In this sense, a multiple-field search approach may allow for better *discoverability* of search features. SAPLE still allows this search method by use of its “Advanced Search” screen, shown in Figure 8.

In SAPLE’s single search box approach, the user’s search query is intelligently interpreted in order to *infer* the query fields and their corresponding *values*. Achieving this in practice requires two capabilities. First, multiple `field:value` pairs must be *parsed* from the user’s query string (for example, `org:5527 type:staff-tech`). The second capability is described below.

Figure 7: Google.com interpretation of multiple query parameters from a single text string input

The screenshot shows the SAPLE Advanced Search screen. At the top, there is a header with the word "saple" in a stylized font. Below the header, there is a section titled "Advanced Search". Under this section, there is a row of buttons labeled "Display Options:", "Picture", "List", and "Include Visitors". Below this row, there are several text input fields, each with a label: "Name or Nickname:", "Organization:", "Phone, Fax, or Cell:", "Building:", "Mail Stop:", "Email ID:", "SNL ID:", "Area:", "Type:", and "Title:". At the bottom right of the form, there are two buttons: "Search" and "Cancel".

Figure 8: SAPLE's Advanced Search screen. The values entered in by the user for each text box are consolidated into a single string with multiple `field:value` pairs, illustrated in Figure 7.

3.1 Inference of Query Terms

The second capability needed to support a single search box approach is the *inference of query terms*. To illustrate, consider the Google search query described above. What if, instead, the search query were `procopio pdf sandia.gov`? In this case, it seems clear that the user had the same or similar intention as the search `procopio filetype:pdf site:sandia.gov`; it can be inferred that the user means to constrain the search to PDF files. It is less clear whether or not they mean to constrain their search to only PDF files hosted on the `*.sandia.gov` website, or if they are looking for PDF files that contain both `procopio` and `sandia.gov`. This is representative of the challenges and ambiguities that can occur when attempting to perform automatic inference on a single-box search query.

In SAPLE, where possible, field names are inferred if only corresponding *values* are provided; so `5527 staff-tech` will be inferred as `org:5527 type:staff-tech`. This is one of SAPLE's most important features in support of efficient and accurate personnel searching. The aim is for the user to be able to specify whatever information they know about a person. As another example, if a Sandian met someone named Mike "P-something" while walking to a meeting in Area 1, they could search `Mike P tech staff TA1`. Here, SAPLE would know that "tech staff" refers to an employee type, and "TA1" refers to a work area, and hence the interpreted query would be `Mike P jobtitle:tech staff workarea:TA1`.

Figure 9 presents an example of SAPLE's automatic query interpretation. Here, the query `M Proco 5527 845-9653 Employee 758 tech staff TA1` is automatically interpreted as `M Proco org:5527 phone:845-9653 employeeetype:Employee building:758 jobtitle:tech staff workarea:TA1`. In a very important usability feature, SAPLE's interpretation of the user query is always rendered back to the user so that they may confirm the automatic interpretation or make adjustments accordingly. In this example, organization, phone number, employee type, building job title, and work area were all automatically parsed and interpreted, leaving the remaining query terms, "M Proco," for the actual name component of the search.

SAPLE is able to perform field name inference for many values through a careful partitioning of the input space. For example, numerical values of seven digits or larger is probably a phone number; three to six digits is (likely) an organization number. However, such a number could also be a building number of a mailstop number. Such *collisions* in the input (or search) space demonstrate how challenging it can be to reliably automatically interpret certain types of queries.

The screenshot displays the SAPLE search interface. At the top, a search bar contains the query "M Proco 5527 845-9653 Employee 758 Tech Sta". To the right of the search bar are links for "Download as Text", "CSV (Excel)", and "Display Options" with buttons for "Picture", "List", and "Include Visitors". Below the search bar, the system has interpreted the query: "Searched for M Proco org:5527 phone:845-9653 employeeetype:Employee building:758 jobtitle:Tech Staff workarea:TA1". The results show "Results 1 - 1 of 1".

The first result is for Michael Procopio ("Mike"). It includes a profile picture and a table of contact information:

Michael Procopio ("Mike")		Details
Email	Office (505) 845-9653	Location
	Fax (505) 284-6231	
STAFF-TECH	Mobile	User ID
Computer Software Rsrch/Dvlpmnt		
Org 5527 Next Generation Monitoring Sys		MySite Profile 5527 Manager

At the bottom of the page, there is a footer with the SAPLE logo, a "maintained by SWIFT!" logo, website contact information for SWIFT and CCHD, and copyright information for Sandia Corporation and Lockheed Martin Corporation.

Figure 9: Example of a compound query (i.e., query with multiple `field:value` pairs)

3.2 Cautious Query Inference

To cope with the above challenge, we take a number of approaches. First, some fields are not automatically inferred; for example, mailstops are never inferred and so must be used with the `mailstop:` or `ms:` field name prefix. For the same reasons (potential collisions with buildings, orgs, etc.), Sandia IDs (or SNL IDs) are never inferred and so must always be prefixed with the field name (`snlid:71569`).

Sometimes, a potential collision is identified, but we can be reasonably sure of the user's intention. For example, the query `6710` probably refers to Organization 6710 (since, as shown in the analytics in Section 6.2, organization queries form a larger percentage of SAPLE searches than building queries). In this case, an actual search for `org:6710` is inferred and performed, but the user is presented with an option to run their search as a building search, as shown in Figure 10. This is a data-driven design decision.

The third primary mechanism for performing automatic query inference is to use an *alias lookup table*, which identifies certain search terms as values that can only be paired with a specific field. For example, when `TA1` is encountered, it is natural to infer that the user means `workarea:TA1`. `area 1` is similarly associated with the `workarea` prefix (field name), i.e., it is an alias for `TA1`. The same general example applies to job titles; the terms `manager`, `mgr`, and `level 1 mgr` can all be considered aliases and interpreted as `jobtitle:manager`. Table 2 provides an condensed example of the alias lookup table for the `job title` field.

Care must be taken so as to not automatically associate a legitimate query component for the name portion of a query with a value keyword. In the above situation, it seems unlikely that a user searching on *manager* would actually mean to search for someone *named* manager. (However, this query interpretation behavior can be overridden by specifying the `name:` prefix, i.e., `name:manager`, and people named “Mansker” and “Manzer” will show up.) This situation illustrates the fine line between maximizing automatic interpretation accuracy and convenience for the user, while not misinterpreting their intention. For example, for the Nevada Test Site, `NTS` would be a useful query that could reliably interpreted as `worksite:nts` and so it is considered an alias. On the other hand, using “Nevada” as an alias would be bad, since there are in fact staff at Sandia with that last name and somebody searching for “Ernie Nevada” would be surprised to get a listing of all staff at the Nevada Test Site instead!

Figure 10: Example of a “search space collision,” in which an ambiguous value (not specifically associated with a field name), if the field name is *inferred*, could be associated with multiple fields. In this case, the user query `6710` could be in reference to both Building 6710, or Organization 6710, both of which are real entities.

Table 2: Example Alias Table for Job Title (plural entries removed for brevity)

Title	Alias	Alias	Alias	Alias
MANAGER	MGR	Level 1 Manager	Level 1 Mgr	
Ombuds	Ombudsman			
POST DOC	Post Doctoral	Postdoc		
SR ADMIN	Senior Admin	Senior Administrator		
SR MGR	Senior Manager	Level 2 Manager	Level 2 Mgr	Sr Manager
STAFF-LAB	Lab Staff	Laboratory Staff	Stafflab	Labstaff
STAFF-TECH	Tech Staff	Technical Staff	Stafftech	techstaff
STUDENT INT	Intern	Student	Student Intern	

3.3 Implementation of Query Parsing

In implementation, SAPLE uses a sophisticated *regular expression matching* scheme, in which tokens of the query are matched to their most likely field names, with preference being given to longer string matches and certain hard-coded priorities where applicable. Such an approach makes case-insensitive and plural matching generally easier, and the matching itself is extremely reliable once the intricate regular expression *patterns* are designed and tested.

SAPLE also supports limited natural language search, by using a simple, naive scheme for selective *removal* of certain tokens. For example, a select list of verbs such as “show all,” “search for,” and “return” are pruned behind-the-scenes from the user’s search. Similarly, quantifiers appearing after a verb such as “all” and “only” are removed, as are certain prepositions. The result allows for queries such as **show all tech staff at snl/ca in center 8600** to be successfully evaluated, although log analysis shows that natural language queries are not common.

4 Scoring and Ranking Search Results

At the very core of SAPLE lies its ability to return a set of personnel results matching the user’s query, with the constraint that the top-most results are the highest ranking, that is, best scoring. Implicit in this general approach is that *each person* in SAPLE’s personnel index must be *scored* with respect to the user’s query. The scores are sorted, with the resulting ranked list of results returned to the presentation layer for display to the user. This general approach is analogous in principle to that taken by other search engines.

SAPLE’s scoring mechanism comprises two parts, and the exact scoring operation will depend on the nature of the search query. As discussed in Section 3, SAPLE handles compound queries, which can comprise both *name tokens* and *filters*. Name tokens refer to those search tokens (components of the search string) that are intended to match part of a name—first, last, nickname, etc. These tokens are used by the search algorithms (discussed below) to score matches in SAPLE’s personnel index.

In contrast, a filter is designed to exclude certain results altogether. For example, in the query **mike percoppia tech staff**, “mike” and “percoppia” are name tokens, while “tech staff” is a filter that constraints the results to only those individuals whose employee type is STAFF-TECH. Other filters include organization filters (**org:5527**), building filters (**building:758**), mailstop filters (**ms:0401**), and so on.

In practice, filters are evaluated first, because they are computationally inexpensive to evaluate. (They prune the number of results that must next be considered by the more expensive approximate string matching algorithms.) Because SAPLE supports multiple filters, filters can be combined together in a LOGICAL AND operation such that results that do not match all specified filters (e.g., Filter A AND Filter B AND Filter C) will be excluded from the results.

After evaluating any filters, each of the remaining results is scored by SAPLE’s search algorithms, using any name tokens provided in the search query. If there are no name tokens provided, for example, if the search query only consists of filters as would be the case for **Division 5000 directors**, then all of the results that do not get excluded by the filter will be returned, and will have equal score. By default, results with equal score will simply be returned in alphabetical order, last name first.

If one or more name tokens is provided, however, SAPLE will score each result in the personnel index (i.e., any ones not already excluded by filters) using a series of search algorithms described below.

4.1 SAPLE Fuzzy Search Algorithms

When performing approximate string matching, SAPLE adopts an *ensemble* approach, in which the similarity scores from *multiple* algorithms are considered. Further, some algorithms carry more weight than others, so the resulting combination of the various algorithms’ scores is done in a weighted manner.

The algorithms used within SAPLE to score individual personnel index entries against the user query are presented below. These algorithms are divided into two general categories: String Similarity Algorithms, and Phonetic Algorithms. (See Section 1.2 for additional background on these algorithms.)

String-Similarity Algorithms

- **Edit distance**, see [2]; also known as Levenshtein distance. This algorithm calculates the minimum number of replacements, substitutions, or deletions required to make two words exactly the same. This value is normalized by a hand-picked maximum value (here, 10), and finally is subtracted from 1 to arrive at the final score. Thus, scores will range on the interval $[0, 1]$, with higher scores representing “closer” (more similar) words. As an example, the edit distance to go from “apple” to “ape” is two deletions. Here, the final similarity score will be $1 - (2/10) = 0.8$, which is higher than the score for “apple” to “apricot” indicating that the former pair is more similar than the latter.

- **Prefix.** A simple, special-use algorithm that rates the length of the prefixes that two words have in common relative to the length of the shortest of the two words. This is accomplished by dividing the length of the common prefix by the length of the shorter of the two words. For example, “catastrophe” and “cataclysm” have a common prefix of “cata” with length 4 out of a possible 9 because “cataclysm” has only 9 letters. The similarity score for these two words is thus 4/9.
- **Containment.** This is a simple, special-use algorithm that checks if the search term is contained in a name from the index. Being contained at the beginning of the word is preferred by being scored a 1 while being contained anywhere else is scored as 0.5. A lack of containment is given a zero. For example, the similarity score using containment of “dog” in “doghouse” will be scored a 1 while the containment of “house” in “doghouse” will only receive a 0.5. For certain cases, this behavior duplicates that of the Prefix algorithm described above. This illuminates the more general challenge of an ensemble-of-algorithms approach; some algorithms will yield similar results for certain cases, but different results in others, and can thus be said to *overlap* in some areas of input space.

Phonetic Algorithms

- **Double Metaphone**, see [13]. Calculates a phonetic encoding of two words according to the Metaphone algorithm, and checks if they are exactly the same. If so, the similarity score is 1; otherwise, it is scored as zero. Double Metaphone focuses mostly on consonant sounds, like many other phonetic encodings. This is based on the observation that as a group, vowel sounds are more easily confused for each other in conversation and especially across dialects. Moreover, vowels can have widely varying pronunciation depending on context (“said” and “bread”, “said” and “wait”). Because of this, the Double Metaphone algorithm only considers the vowels that are the first character in the word and even then, all vowels receive the same encoding.
- **NYSIIS**, see [14]. The New York State Identification and Intelligence System phonetic code is a simpler encoding and with the same scoring strategy as Double Metaphone (score of 1 if encodings match exactly, score of 0 otherwise). A custom set of encodings is present; for example “ph” and “pf” both map to “ff” in the encoding string. The encodings are stored as 32-bit integers, which limits the length of the encoding and has the potential to destroy later evidence in the source words. This bias towards considering later characters less is present in other similarity algorithms, however.
- **Phonix**, see [6]. Phonix is similar encoding and scoring strategy to Double Metaphone and NYSIIS. Here, encodings are stored as 16-bit integers. The set of encodings shares similarities but also has differences from the other phonetic encoding methods above. For example, “b” and “p” are equated, as are “f” and “v.” One can see how these consonant sounds could easily be confused in conversation, in particular with last names.
- **EDITEX**, see [19]. Editex is combination of both edit distance and phonetic similarity. As with the Levenshtein algorithm, a basic edit distance is computed, identifying the requires substitutions, deletions, and swaps required to transform one word to another. However, the costs (distances) incurred for deletions and replacements are no longer constant; instead, they are a function of the phonetic properties of the letters in question. For example, swapping “p” and “b” should incur lower cost than swapping “p” and “l” because the former is more phonetically similar.

Other algorithms considered, but not used

Some algorithms were implemented for SAPLE and were, at one point in time, used within SAPLE. They have now been excluded due to one of three reasons. First, some algorithms give poor performance in terms of their impact on search quality (Soundex). Second, some algorithms were excluded because their output was very similar to that of another algorithm already being used (Longest Common Substring/Subsequence). Finally, some algorithms performed well, but their computational requirements were high, to the point of causing unacceptable latency (i.e., low response times) in returning search results (N -gram).

4.2 Combination of Algorithms

The algorithms are executed independently, with each returning a match score. The resulting scores given by the algorithm ensemble are combined on a per-word basis as a weighted average of all the algorithm scores. That is, each algorithm is associated with a weight, with final combination for a particular name part (first name, last name, nickname, etc.) being a linear combination of the algorithm’s score and its associated weight. The composite score for the entire name is calculated as the product of the squares of the per-component scores. In this sense, the best score is a 1 (first/middle/last name having a combined algorithm score of 1, i.e., exact match), while the worst score is 0.

As of this writing, the algorithm weights have been hand-tuned by the developers after extensive testing and observation. Editex and Edit Distance (Levenshtein) receive the majority of the weight in the algorithm ensemble. The current weights are:

- Edit Distance: 0.3
- Prefix: 0.1
- Containment: 0.05
- Double Metaphone: 0.1
- NYSHS: 0.1
- Phonix: 0.05
- Editex: 0.3

4.3 Examples of Algorithm Ensemble Combination

Tables 3 and 4 provide specific examples for how scores are computed in the above weighted ensemble of algorithms. In Table 3, consider a user’s query for the last name “Tayler,” which is a misspelling and no such last name exists in the Sandia personnel database. Phonetically, the user probably means “Taylor.” On the other hand, the extra “a” could be a typographical error.

So, “Taylor” and “Tyler” then seem to be reasonable matches. The key question is: what should their relative ranking be in the search results? Relative rank is the key question that must be answered by any search engine. As described above, SAPLE takes the weighted linear combination of multiple algorithms, including both string-similarity algorithms and phonetic algorithms.

In Table 3, the final score (represented as the sum of the weighted terms (the *weighted combination*) for “Tayler” vs. “Taylor” is shown as the left number in the last row (0.86). The score for “Taylor” vs. “Tyler” is shown as the right number (0.78). Because the score for “Tayler” vs. “Taylor” is higher than vs. “Tyler,” the “Taylor” entry would be ranked higher. What accounts for the difference? Most of the algorithms rated these two candidate matches the same. However, “Taylor” was a closer match than “Tyler” for the Prefix algorithm, because the common prefix “Tayl” has 4 characters, normalized by a word length of 6, for a score of 0.67. This is in contrast to “Tyler,” where the shared prefix “T” only has one character, normalized by 6, is 0.17.

Meanwhile, while both had the same Edit Distance score, Editex says the replacement of “e” with “o” to make “Tayler” be “Taylor” *costs less* than the deletion of “a” to make “Taylor” be “Tyler.” With everything else a tie, “Taylor” has a higher score because of these two algorithms rating it slightly higher. Note that an interesting situation arises here; “Tyler” is both a last name and a first name. By convention, when there is a tie, precedence in the ranking is given to last name. Therefore, everything else being equal, “Tyler” will result in results with people with last name “Tyler” ranked higher than those with first name “Tyler.”

Consider another example in Table 4. Here, the user searches for the last name “Consillo,” where perhaps they meant “Cancilla.” Another reasonable match could be “Castillo.” Here, “Cancilla” (final score of 0.58) will be ranked higher than Castillo (final score of 0.37). As above, the Edit Distance algorithm reports the exact same score, because in *both* cases, three letters must be replaced make “Consillo” match. However, the Editex algorithm’s ability begins to show here. It treats a replacing vowels as less expensive than replacing most consonants. Replacing the “ons” in “Consillo” with “can” to make “Castillo” is more expensive, phonetically, than replacing two “o” characters in “Consillo” with “a,” and one “s” character with “c,” to make “Cancilla.” This seems very intuitive.

The Double Metaphone algorithm also identifies the same phonetic encoding for “Consillo” and “Cancilla” (*KNSL*) but not for “Castillo” (*KSTL*). The same applies to the Phonix. Overall, in the final weighted combination (sum), “Cancilla” has a higher score and will be ranked higher than “Castillo.”

Table 3: Score computation table for user query “Tayler” vs. candidate matches “Taylor” and “Tyler”

USER QUERY: “Tayler”					
ALGORITHM	WEIGHT	vs. “Taylor”		vs. “Tyler”	
		SCORE	WEIGHTED	SCORE	WEIGHTED
Edit Distance	0.30	0.90	0.27	0.90	0.27
Prefix	0.10	0.67	0.07	0.17	0.02
Containment	0.05	0.00	0.00	0.00	0.00
DoubleMeta	0.10	1.00	0.10	1.00	0.10
NYSIIS	0.10	1.00	0.10	1.00	0.10
Phonix	0.05	1.00	0.05	1.00	0.05
Editex	0.30	0.90	0.27	0.80	0.24
WEIGHTED COMBINATION (SUM)			0.86	0.78	

Table 4: Score computation table for user query “Consillo” vs. candidate matches “Cancilla” and “Castillo”

USER QUERY: “Consillo”					
ALGORITHM	WEIGHT	vs. “Cancilla”		vs. “Castillo”	
		SCORE	WEIGHTED	SCORE	WEIGHTED
Edit Distance	0.30	0.70	0.21	0.70	0.21
Prefix	0.10	0.13	0.01	0.13	0.01
Containment	0.05	0.00	0.00	0.00	0.00
DoubleMeta	0.10	1.00	0.10	0.00	0.00
NYSIIS	0.10	0.00	0.00	0.00	0.00
Phonix	0.05	1.00	0.05	0.00	0.00
Editex	0.30	0.70	0.21	0.50	0.15
WEIGHTED COMBINATION (SUM)			0.58	0.37	

5 SAPLE Web Service

Across the laboratory, software developers have written a number of “home grown” personnel searches, each custom-developed for one particular web application. A more efficient and cost-effective approach would be to provide a centralized directory search service, implemented and made available as a web service for other applications to use. Then, the various personnel search components across Sandia could be consolidated. As a result, the software development and maintenance burden could be greatly reduced, while the user experience—the person using the directory component in the web application—would be more uniform.

Further, to the extent that the centralized directory web service provides a more effective or easier to use search capability, those benefits would accordingly be passed on to any client applications (both web and desktop applications) making use of the SAPLE search service. Finally, using a centralized search service with a single, clean interface encapsulates and hides a large amount of infrastructure complexity, freeing the application developers from having to do any personnel database queries, custom string parsing, and so on.

5.1 Web Service Interface and Specification

SAPLE was designed to act like such a service; in fact, it could be said that the SAPLE web application itself (at <http://saple.sandia.gov>) is just one example of many customized end-user applications making use of, or *consuming*, the services provided by the SAPLE search engine.

SAPLE’s web services implementation is very simple, and is designed to allow very quick adoption with minimal overhead in a software engineering context. SAPLE adopts a *RESTful* web services approach, which does not expose a formal set of methods or RPC (remote procedure call) specification. Moreover, this approach does not require software overhead associated with more sophisticated web services implementations. Such implementations may expose a WSDL (Web Service Definition Language) file, and may require the use of SOAP (Simple Object Access Protocol) to transmit search queries and receive results.

In contrast, the SAPLE web service has a single URL-based specification to invoke a search: <https://webprod.sandia.gov/SAPLE/search?query=<user’s query here>>. For example, one would use the URL <https://webprod.sandia.gov/SAPLE/search?query=procopio 5527> to use the web service to search for staff members named “procopio” in Organization 5527. This in turn invokes the search *servlet* and the search otherwise follows the same execution path inside the application as a search submitted through the web interface (i.e., using the `search.jsp` file).

SAPLE returns search results in XML format. These results conform to a well defined specification, provided to the user as an *XML Schema* file. This file can be accessed by anyone, and the URL to the file is provided in the XML set of results returned in response to a query. The XML data returned is guaranteed to *validate* against the XML Schema. Software developers can then develop their applications against this specification. Figure 11 provides an example of the XML results as returned by the SAPLE web service, as well as a snapshot of the XML schema specification to which the results conform.

Access to the SAPLE web service must be done in an *authenticated context*. Application developers making use of the SAPLE web service are encouraged to create an *entity account*, which will be granted access to the SAPLE web service once that entity account is added to the `wg-saple-entityacct` *metagroup*.

Figure 11: Example XML result set from the SAPLE web service (left). XML Schema Excerpt (saple.xsd) (right).

5.2 Example Client Application: Software Asset Management System (SAMS)

The first external application to use the SAPLE web service is the Software Asset Management System (SAMS) application, which provides real-time, fully automated software acquisition and installation at Sandia. SAMS provides software license tracking and an eCommerce-like online store, where customers acquire, install, uninstall, and transfer software. Sandia personnel play a prominent role in this application, because some software licenses are tracked at the individual level. Instead of custom-developing a one-off personnel lookup component from scratch, the SAMS application developers simply connected to the SAPLE Web Service.

All of the features of SAPLE—single-textbox query interpretation, inexact string matching, compound query searches, etc.—are fully available to the SAMS application user. The SAMS software developers use AJAX (asynchronous JavaScript and XML) to create a rich user experience, passing the user’s query to the SAPLE service, parsing the returned XML result set, and dynamically presenting the results to the user in a drop down list. Figure 12 shows SAPLE in use by the SAMS application.

Figure 12: Screenshot of SAMS Application. In this scenario, the query `org 10501 lab staff` has been passed to the SAPLE web service via AJAX (JavaScript), and the XML result set returned by SAPLE has been parsed and presented to the user in a dropdown list.

6 Analytics and Usage Analysis

SAPLE, by design, logs every search query performed in order to support extensive *search analytics* later on. Such analytics can provide useful insights into how SAPLE is being used, and drive priorities for future enhancement. For example, for the static analysis presented here, it was discovered that a small but significant percentage of SAPLE search queries failing—i.e., those queries returning zero results—were due to searches like `mike 55` and `57 managers`. Inspection shows that the user intended for the two-number search term to be interpreted as the Center, i.e., Center 5500 and Center 5700. A small change in the SAPLE program code was made and now the user is prompted with a “Did you mean: *center 5700* managers” for a query such as `57 managers`. Similar analysis yielded search quality improvements for queries containing three-digit organization numbers.

As described in the above scenario, log file analysis can be done on a one-time basis, but such analysis can also occur on an ongoing basis, dynamically and in real-time, in order to automatically tune the search system to yield increasing search quality over time. For example, the weights used in the linear weighting scheme for combining the ensemble of algorithm scores (Section 4.1) could be tuned over time, if it was found that one of the algorithms in the ensemble was more correlated with successful search. (This has been done manually by inspection in the past, but such hand-tuning can be fragile and time consuming.) Such dynamic or *on-line* tuning of the system is not currently in place, but could be an element of future work.

In order to quantify and characterize SAPLE usage patterns, an in-depth, one-time analysis was performed. The following sections present key results and statistics from logfile analytics performed on SAPLE query data during a sample period of 9/21/2009 to 12/4/2009.

6.1 General Usage and Load

One of the first questions that can be asked is, “how much is SAPLE used?” For the sample period (9/21/2009 to 12/4/2009), SAPLE was used to conduct 515,272 searches. Excluding the Thanksgiving holiday week, this represents an average weekly load of about 48,727 searches. On standard business days, SAPLE received about 10,687 queries per day on average during the sample period. On Fridays, as expected, the search load was about half that, 5,292. (This is because at Sandia, a large number of staff work a compressed work week in which every other Friday is taken off.) A chart showing the usage, broken out by day and grouped by week, is shown in Figure 13.

Figure 13: SAPLE Usage for Monday, 9/21/2009 – Friday, 12/4/2009.

Figure 14 characterizes the search load for one day, Thursday, 12/3/2009. SAPLE begins to see searches around 6:00am, with the search load peaking between 9am and 10am, falling off slightly towards the middle of the day (presumably to staff’s lunch period), resuming at slightly reduced rates in the afternoon, tapering off around 4pm, and finally reducing to very nominal levels after 6pm. The heaviest search load appears to be at a rate of approximately 1500 searches per hour, or about one search every two seconds. This corroborates estimated search loads from a survey of Sandia staff familiar with the legacy directory application, and is well within target search loads conducted in load testing of approximately 3,000 searches/hour.

Figure 14: SAPLE Usage for Thursday, 12/3/2009.

6.2 Search Query Type Breakdown

The breakdown of search queries processed by SAPLE during the sample period is given in Table 5. Approximately *half* of the searches performed (49%) consist of a single token name search, for example, `martinez` or `gardner`. Half of the remaining searches (25% total) are organization searches, including Organization, Business Area, Center, and Division. 20% of the total searches are name searches involving two tokens (e.g., `last_name, first_name`). The remaining 5% of the searches include searches for building, phone number, mailstop, User ID, and related queries, as well as *compound* queries containing multiple query components (e.g., `Mike center:5500`).

Table 5: SAPLE Searches by Type

QUERY TYPE	COUNT	% OF TOTAL
name (single token)	263899	49%
org	132657	25%
name (two or more tokens)	102747	19%
userid	24040	4%
other (bldg, phone, etc.)	12982	2%
compound searches	3690	1%

^a *Other* category includes building, phone, jobtitle, mailstop, type, snlid, area, and room.

Figure 15: Breakdown of SAPLE Searches by type. Type names correspond to entries in Table 5. The data show that single-token name searches and organization-related searches form the majority of the queries processed by SAPLE.

Top 50 SAPLE Queries

The top 50 most frequent SAPLE queries are listed in Table 6. These 50 distinct query terms represent approximately 5% of the total searches considered in the sample period, which is not particularly high; the query space handled by SAPLE is quite large. Nonetheless, looking at this data can provide some interesting insights into how SAPLE is used, which are confirmed in later analysis. Not surprisingly, the most common searches in SAPLE are single-token last name searches. Names that are more heavily represented in the Sandia population, such as martinez, chavez, and smith, generally receive proportionately higher search density.

Observation yields that searches on Center (e.g., 5400) are also very common, although this can be explained somewhat by considering there is a relatively small number of Centers at Sandia (approximately 50 to 60). A second factor that may contribute to this is the increased use of SAPLE to dynamically generate organization rosters; in multiple places throughout the labs, department homepages are linking to SAPLE to show listings at the Organization, Business area, and Center levels.

200 Random SAPLE Queries

Table 7 lists a random sample of 200 queries from the SAPLE logs during the sample period. This gives a representative snapshot of the sorts of things that SAPLE is used to search for, as well as a rough point estimate of the frequency of certain types of searches (this is characterized fully in later analysis). The data comprise a variety of search types, from single-token last name searches (both wildcarded and not), org searches, phone number lookups, User ID lookups, job title lookups, etc. Rarely, a non-personnel related query, such as “video services,” shows up in the data; this indicates people, through one mechanism or another, are on occasionally using SAPLE to search for something it is not designed to search for.

Table 6: Top 50 SAPLE Search Queries^a

RANK	QUERY	COUNT	RANK	QUERY	COUNT
1	martinez	903	26	5400	450
2	2700	898	27	anderson	449
3	4211	777	28	trujillo	447
4	chavez	768	29	4000	439
5	smith	763	30	gonzales	437
6	garcia	753	31	6000	418
7	sanchez	674	32	org:9343	418
8	5700	673	33	5600	415
9	romero	660	34	1700	414
10	6700	610	35	6400	412
11	4100	595	36	lopez	411
12	5000	569	37	1500	405
13	jones	567	38	5300	404
14	montoya	548	39	5900	393
15	johnson	536	40	10500	392
16	lucero	523	41	10200	378
17	4200	510	42	gardner	368
18	davis	508	43	padilla	363
19	baca	508	44	4800	361
20	williams	489	45	2900	357
21	plummer	485	46	rivera	349
22	2600	477	47	clark	347
23	millar	466	48	gallegos	344
24	2000	453	49	walker	340
25	brown	451	50	moore	338

^a Top 50 queries represent 4.7% of total searches.

Table 7: Random 200 SAPLE Search Queries

anaya	org:2110	org:5579	kempka
shiplet	8446067	userid:drhelmi	eppinga
millers,mark	org:00421	orsbun	coffey
4219	martin	den rid	goodrich
userid:jamaher	phone:284-6327	video services	djwilco
sanch	sinclair	10600	nelson,sh
hill	evantich	curtis	spence
amberg	falconi	business area:5340	5700
lon	gutierrez, t	slater	steve
kowalchuk	division:5000	org:8622	lo
harper	smith	robertson, donna	chavez
lovato-teague	alexander, richard	userid:sakemme	petersen
boruff	10503	org:9753	chen
felix	Anderson	102633	dorsey
cook	ewen	ross,k	gallegos,john
2700	eisenman	romero, mary	hall, g
org:00212	lau	glover	lovato
flores,c	anderson, d	dramer	stubblefield
grach	drayer	johnsen	userid:dmartin
neely	284-9986	org:6451	barclay
McLain	4211	chri	org:1813
Hogue	sanchez-godin	farmer	pitman
hale	5900	org:1516	STAKE
kim gallagher	paz, l	4225	blackar
org:6736	hertel	phone:5058450130	9383
vega-prue	mckean,	solis, p	ayer
gallegos, grace	montoya, charle	Johnson, Alfred E	koch, consta
jensen	werhling	userid:wrbolto	Lingerfelt
2733	5343	polonis	5350
lopez,jose	Lu	5900	witze
userid:bahendr	mcdowel	phone:8446568	sena
oneil	hartman	%Janet	4217
2662	ANDREW LOPEZ	Hardesty	glover, b
org:5353	Wakefield	org:1526	brightwell
barbour	org:42181	heckart	hamilton, tom
chase	mead,m	preston	mcilroy
martinez	may, ro	BARNETT,	GRIFFITH
schorzmamm	gunckle	phone:8457139	phone:8450555
delozier	Naranjo	6330	cook, j
Duus	center:6300	rondeau	10629
wheeler	stayton, ann	org:6338	Pickering
rogers	drew parson	Heyborne,Melanie	1800
phone:5052841246	anderson, robert	Rom	shaver
4211	gall	org:8621	holle
1800	mert	Mulligan	10263
garcia, p	Montoya	pope, r	sullivan
lopez, mich	org:9537	trujillo	4848
walck	webb	garcia, virginia	lierz
userid:wdwilli	6760	title:MANAGER	1832

6.3 Search Usage by User

During the sample period, SAPLE was used by 9,936 unique users, which represents the majority of the people with access to the search mechanism at the laboratory. The distribution of the searches was not uniform across users, as shown in Figure 16. Some users had very heavy search loads (with six users with over a thousand searches during the sample period). Others had more moderate loads, with half the users represented in the population had 16 searches or less. Overall, the distribution exhibits an approximately logarithmic trend.

Figure 16: SAPLE search usage (frequency) for the top 5,000 SAPLE users

6.4 User Type Breakdown

Table 8 presents a full breakdown of SAPLE usage by user *type*, i.e., job title. Table 17 presents a consolidated breakdown, with the resulting distribution charted in Figure 18. Overall, SAPLE usage is distributed quite evenly across the various job titles across the laboratory. Management, technical staff, laboratory staff, and administrative staff are all well represented.

A secondary analysis was performed in order to compare the distribution of SAPLE user types with the overall distribution of Sandia’s general population. This analysis revealed that overall, Sandia technical staff (including technologists) used SAPLE commensurate with their representation in the general population. However, management, administrative staff, laboratory staff, and student interns all used SAPLE approximately three times *more frequently* than their proportion in the general population would predict. In contrast, contractors, including staff augmentation, used SAPLE less three times *less frequently* than their representation in the general population.

Table 8: SAPLE Searches by Title (All Titles)

TITLE	COUNT	%	TITLE	COUNT	%
STAFF-TECH	129596	24.0%	GRP TRADES	4376	0.8%
CONTRACTOR ^a	75048	13.9%	LTD TERM	4208	0.8%
STAFF-LAB	70694	13.1%	DIRECTOR	2458	0.5%
MANAGER	48445	9.0%	POST DOC	1883	0.3%
OAA	41782	7.7%	SECURITY	1854	0.3%
TECHNOLGIST	41322	7.6%	GRADED	1814	0.3%
ASA	38779	7.2%	TRADES	956	0.2%
SR MGT AST	18138	3.4%	SR SCI/ENG	833	0.2%
MGMT ASST	17113	3.2%	SR ADMIN	831	0.2%
TIER	14849	2.7%	TEAM SUPVR	610	0.1%
SR MGR	9495	1.8%	TEAM LIEUT	332	0.1%
STUDENT INT	8469	1.6%	VICE PRES ^b	378	0.1%
TEAM LEADER	5536	1.0%	OTHER ^c	1004	0.2%

^a Includes Staff Augmentation.

^b Includes VP-EXEC.

^c Includes TEAM CAPT, OMBUDS, DSP/MFP, etc.

Figure 17: SAPLE Searches by Title (Grouped)

TITLE	COUNT	%
STAFF-TECH	129596	24.0%
ADMINISTRATIVE ^a	116643	21.6%
CONTRACTOR ^b	75048	13.9%
STAFF-LAB	70694	13.1%
MANAGEMENT ^c	64684	12.0%
TECHNOLGIST	41322	7.6%
TIER	14849	2.7%
STUDENT INT	8469	1.6%
EXECUTIVE ^d	2836	0.5%
OTHER ^e	16576	3.1%

^a Includes OAA, ASA, (SR) MGT ASST, SR ADMIN

^b Includes PO Contractor, Staff Augmentation.

^c Includes MANAGER, SR MGR, TEAM LEADR, etc.

^d Includes DIRECTOR, VICE PRES, and VP-EXEC.

^e Includes GRP TRADES, LTD TERM, GRADED, etc.

Figure 18: Breakdown of SAPLE Searches by Title

6.5 SAPLE Search Sources

Table 9 lists the breakdown of SAPLE search *sources*, that is, where the SAPLE query is initiated. Currently, the majority (69%) of SAPLE queries are initiated from Techweb (the search textbox in the upper-right hand corner of the Techweb page). About 24% of the searches came from the SAPLE search *header*, the search text box at the top of a SAPLE results page. Most of these are likely follow-up queries to the original query. About 2.5% of searches came from the “Advanced Search” page (see Figure 8), with another 2.5% coming from the SAPLE homepage (<http://saple.sandia.gov>). About 1% of SAPLE searches were done from the search footer (the search text box at the bottom of a SAPLE search results page, suggesting that it could be eliminated with minimal impact while making the user interface a little neater. Another 1% of searches came from external applications; as of this writing, these are searches initiated from the SAMS application (see Section 5.2).

SAPLE also tracks *internal* actions, which are not a user-typed query directly, but do represent a user action; the bottom half of Table 9 provides the breakdown. There are a number of important inferences from this data. First, users frequently click on the “Org Roster” listing for a specific entry. So if they were searching for Joe Smith, about 10% of the time, they want see the other members of Joe’s organization. This is an important finding.

The Manager Lookup function was also clicked frequently; this may be used less frequently with the recent changes of showing the Manager of Record and Secretary of Record at the top of an organization listing page (see Figure 2b). Another key finding here is that the “Next” link—which advances to the next page of results—is clicked extremely infrequently, less than 0.1% of the time. This is strong evidence that the majority of the time, the user has found who they are looking for on the first page. A counter argument to this claim, however, could be that the high number of Search Header searches indicate that some percentage of the time, users are submitting revised queries directly from a results page.

Finally, the “Show all” function is almost never clicked. If it is useful, it probably is not very discoverable. Regardless, it could probably be safely eliminated with minimal user impact.

Table 9: SAPLE Search Sources

SOURCE	COUNT	%
SAPLE Search Query Sources, External		
Techweb	319406	68.9%
Search Header	113123	24.4%
SAPLE Home Page	12135	2.6%
Advanced Search	10945	2.4%
Search Footer	4625	1.0%
External Applications	3485	0.8%
SAPLE-Internal Search Actions		
Org Roster	58808	—
Manager Lookup	16368	—
“Did you Mean” Suggestion	1488	—
Next/Page N	323	—
Show All	12	—

6.6 SAPLE Computational Performance

Some of the search algorithms used by SAPLE (described in Section 4) are very computationally intensive. Moreover, for each query, there are a large number of comparisons that must be done (approximately 20,000 personnel entries in the search index), while the search load placed on the SAPLE application by users is considerable (up to one search a second during peak times). Because the application must be responsive to the user—the target metric is for results to be returned within two seconds, 80% of the time—heavy emphasis has been placed on algorithm optimization.

The following analysis considers the computation time that SAPLE spends on the algorithmic scoring portion of the query. (SAPLE times each search and logs the elapsed time along with other query information.) These times will be exclusive of other overhead associated with SAPLE, for example, the time required to render the JSP search page and to transmit the resulting HTML back to the user’s web browser.

Table 10 presents the computation time of various types of queries; the times are plotted in Figure 19. All searches have an average search time of less than 0.2s, which is excellent. Single-token name searches involving wildcards are fastest; wildcards, by definition, imply an exact match and therefore the computationally intensive fuzzy-matching algorithms are not invoked for queries containing wildcards. *Filter* searches (see Section 3) are also very fast, because filters such as `org:`, `building:`, `userid`, and similar require exact matching. Phone number searches are somewhat slower, because one phone number must be compared with multiple phone number *fields*, for example, office, fax, pager, and mobile.

The non-wildcarded name searches take the longest; these require the invocation of SAPLE’s inexact (fuzzy) string matching algorithms. An important distinction arises when considering those queries with commas, versus those without. Queries in which a user specifies a comma occur almost four times as often as those without a comma, most likely due to many years’ worth of the Legacy Directory application requiring queries to be in this format. When a user inputs a two-token name search with a comma, SAPLE assumes they are signalling that the first token corresponds to the last name, and the second token, the first name (i.e., `last_name, first_name` format). In other words, the user is specifying the order. Therefore, when scoring, each token needs to only be compared with the corresponding first name or last name as stored in the index; this is very efficient.

In contrast, when the user does *not* specify a comma, the order is unspecified. The two tokens could either be in `last_name first_name` or `first_name last_name` formats. In this case, twice as many comparisons need to be performed, because it is not clear which of the two search tokens match up to which of the first/last names in SAPLE’s personnel index. The reality is a bit more complicated, since middle names and nicknames are also considered, but the principle is still valid. One result from this analysis that merits further study concerns the single-token name search without wildcards. Because this involves fewer comparisons overall, its algorithmic execution time should be faster, but instead, it is slower than the other types searches. This will be investigated.

Figure 20 shows the distribution of SAPLE search times for the sample period. The slowest 1% of the queries were disregarded as outliers (server error, network anomaly, unexpectedly high server load, etc.). The distribution exhibits trimodality, which is correlated with the count (frequency) and associated computation time data presented in Table 10. There is strong peak around 100ms, corresponding to the second- and third-most frequent types of searches, org and two-token-with-comma name search (98ms and 108ms mean search computation time, respectively). A broader peak occurs around 180ms, corresponding to the most frequent type of search, the single-token name a search. Finally, a small peak is present around 45ms, corresponding to a single-token name search using wildcards.

Overall, the vast majority of searches performed by SAPLE during the sample period require less than 0.25s of computation time. This result, which is characterized as excellent performance, is due to index-based approach taken by SAPLE in its architecture, as well as the extensive optimization of search and scoring algorithm implementations within SAPLE.

Table 10: SAPLE Computation times and count by query type

QUERY TYPE	EXAMPLE	COUNT	COMPUT. TIME (MS)
name (single, w/ wildcards)	proco*	3681	46
other ^a	ms 0401	1089	90
userid	mjproco	24040	94
org	5527	132657	98
compound query	mike 758	3690	104
name (two tokens, w/comma)	procopio, mike	78041	108
name (three tokens, w/ comma)	procopio, mike j	2459	108
phone	845-9653	8019	136
name (three tokens, w/ no comma)	mike j procopio	2625	142
name (two tokens, no comma)	mike procopio	19622	153
name (single, no wildcards)	procopio	260218	186

^a Other category includes building, jobtitle, mailstop, type, snlid, area, and room.

Figure 19: SAPLE Algorithm Computation Time by Search Type

Figure 20: Distribution of SAPLE Computation Times, Fastest 99%. Approximately 1% of the slowest queries have been removed as outliers. The distribution exhibits clear trimodality, which is correlated to the actual type of search performed and resulting computational complexity.

7 Conclusions and Future Work

In this paper, we presented the Sandia Advanced Personnel Locator Engine (SAPLE) web application, a directory application for use by Sandia National Laboratories personnel. Its purpose is to return Sandia personnel “results” as a function of user search queries. SAPLE’s mission is to make it easier and faster to find people at Sandia. To accomplish this, SAPLE breaks from more traditional directory application approaches by aiming to return the correct set of results while placing very minimal constraints on a user’s query.

Two key features form the core of SAPLE: advanced search query interpretation and inexact string matching. The former feature permits the user to make advanced, compound queries by typing into a single search field. Where able, SAPLE *infers* the type of field that the user intends, for example, 758 refers to a building, while 5527 refers to an organization. The second feature, inexact string matching, allows SAPLE to return a high quality ranking of personnel results that, while not being exact matches to the user’s query, will be close.

SAPLE also exposes a simple-to-use web service, permitting external applications to leverage SAPLE’s search engine with a minimal amount of overhead. All of SAPLE’s benefits—query interpretation, high quality search results—are immediately realized by the consumers of SAPLE’s web service. Meanwhile, application developers are shielded from the complexities of having to develop and maintain a one-off custom search component, including interfacing to Sandia’s proprietary personnel databases, parsing user input, etc. The SAMS application, Sandia’s first external application to use the SAPLE web service, was described.

This paper also presented an extensive characterization of SAPLE usage for an 11-week sample period. Key results from this analysis showed that SAPLE usage was widely distributed over job titles across the laboratory; SAPLE handles approximately 10,000 queries per day, and this search load is generally evenly distributed throughout business hours; most searches are either single-token name searches or organization searches; and SAPLE is able to perform the vast majority using less than 0.25s of compute time.

Future Work

Future work for SAPLE will focus on three key areas. First, the user interface will continue to improve, incorporating higher and higher degrees of rich-client user interface components implemented using AJAX. This will include integration with Sandia’s forthcoming web-based mapping capability.

Second, additional effort will be spent improving SAPLE’s approximate string matching algorithms, to further improve search quality. For example, specific phonetic encodings will be developed to achieve higher search quality when searching for Spanish-language names (see discussion in Section 1.2). Another option for improved search quality involves using the updated *Metaphone 3* algorithm. Also, the discussion in Section 4.3 suggests a high degree of overlap between the Editex and the Edit Distance (Levenshtein) algorithms. One of these could be removed in order to make SAPLE more computationally efficient, improving latency for the end user while incurring a minimal or negligible impact in search quality.

Meanwhile, the same principles used by the Editex algorithm to compute string similarity could also be applied to better cope with mechanical or typographical errors. In this case, the cost of replacing “d” with “s” is low because they are adjacent on the keyboard. Such algorithms are known and in common use. For example, Apple’s iPhone automatically corrects typing based on an Editex-like algorithm with replacement costs based on keyboard letter proximity. Third, advanced scoring and ranking methods that consider patterns in the logged query data, many of which are off-the-shelf algorithms, could be employed to automatically improve the quality of SAPLE’s search results over time. Related to this, principled metrics such as *precision* and *recall*, in particular the former, have been used to evaluate search engine performance in the informational retrieval domain. These could be used to more rigorously quantify the overall quality of the SAPLE search results.

Finally, the benefits of consolidating the many one-off custom personnel search modules deployed in web applications across the laboratory is clear. Future applications requiring a personnel search component should rely on SAPLE's web service component exclusively, while existing applications could be converted to use the centralized SAPLE search service with a minimum of effort. Consolidating personnel search will not only be cost-effective, but the resulting user experience consistency and high-quality search results from a SAPLE-branded search will improve the user experience across the laboratory, allowing personnel searches to be easier and more efficient wherever they are done.

References

- [1] Peter Christen. A comparison of personal name matching: Techniques and practical issues. In *ICDMW '06: Proceedings of the Sixth IEEE International Conference on Data Mining - Workshops*, pages 290–294, Washington, DC, USA, 2006. IEEE Computer Society.
- [2] William W. Cohen, Pradeep Ravikumar, and Stephen E. Fienberg. A comparison of string distance metrics for name-matching tasks. pages 73–78, 2003.
- [3] Richard Durbin, Sean Eddy, Anders Krogh, and Graeme Mitchison. *Biological sequence analysis: probabilistic models of proteins and nucleic acids*. Cambridge Univ. 1998.
- [4] Ahmed K. Elmagarmid, Panagiotis G. Ipeirotis, and Vassilios S. Verykios. Duplicate record detection: A survey. *IEEE Trans. on Knowl. and Data Eng.*, 19(1):1–16, 2007.
- [5] T. N. Gadd. ‘fishing fore werds’: phonetic retrieval of written text in information systems. *Program: electronic library and information systems*, 22(3):222–237, 1988.
- [6] T. N. Gadd. Phonix: The algorithm. *Program: electronic library and information systems*, 24(4):363–366, 1990.
- [7] Patrick A. V. Hall and Geoff R. Dowling. Approximate string matching. *ACM Comput. Surv.*, 12(4):381–402, 1980.
- [8] Victoria J. Hodge, Jim Austin, Yo Dd, and Yo Dd. An evaluation of phonetic spell checkers. Technical report, Mechanisms of Radiation Effects in Electronic Materials, 2001.
- [9] Karen Kukich. Techniques for automatically correcting words in text. *ACM Comput. Surv.*, 24(4):377–439, 1992.
- [10] Gonzalo Navarro. A guided tour to approximate string matching. *ACM Comput. Surv.*, 33(1):31–88, 2001.
- [11] H. B. Newcombe, J. M. Kennedy, S. J. Axford, and A. P. James. Automatic Linkage of Vital Records. *Science*, 130:954–959, October 1959.
- [12] Beatrice T. Oshika, Bruce Evans, Filip Machi, and Janet Tom. Computational techniques for improved name search. In *Proceedings of the second conference on Applied natural language processing*, pages 203–210, Morristown, NJ, USA, 1988. Association for Computational Linguistics.
- [13] Lawrence Philips. The double metaphone search algorithm. *C/C++ Users J.*, 18(6):38–43, 2000.
- [14] Robert L. Taft. Name search techniques. Technical Report 1, New York State Identification and Intelligence System, 1970.
- [15] Esko Ukkonen. Approximate string-matching with q-grams and maximal matches. *Theor. Comput. Sci.*, 92(1):191–211, 1992.
- [16] William E. Winkler. The state of record linkage and current research problems. Technical report, Statistical Research Division, U.S. Census Bureau, 1999.
- [17] Sun Wu and Udi Manber. Agrep - a fast approximate pattern-matching tool. In *In Proc. of USENIX Technical Conference*, pages 153–162, 1992.
- [18] Justin Zobel and Philip Dart. Finding approximate matches in large lexicons. *Softw. Pract. Exper.*, 25(3):331–345, 1995.
- [19] Justin Zobel and Philip Dart. Phonetic string matching: lessons from information retrieval. In *SIGIR '96: Proceedings of the 19th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 166–172, New York, NY, USA, 1996. ACM.

- [20] Justin Zobel, Alistair Moffat, and Ron Sacks-Davis. An efficient indexing technique for full text databases. In *VLDB '92: Proceedings of the 18th International Conference on Very Large Data Bases*, pages 352–362, San Francisco, CA, USA, 1992. Morgan Kaufmann Publishers Inc.

DISTRIBUTION:

- 5 Mike Procopio
Google, Inc.
2590 Pearl St., Suite 110
Boulder, Colorado 80302
- 1 MS 9533 Mathew Anderson, 0805
1 MS 6323 Alisa Bandlow, 1138
1 MS 6324 Keith Bauer, 1138
1 MS 89441 Brian Byers, 0947
3 MS 8944 Samuel Cancilla, 0807
1 MS 8944 Cara Corey, 0947
1 MS 9544 Leonard Chavez, 1457
3 MS 9530 William Cook, 0807
3 MS 5527 Jesse Flemming, 0401
1 MS 5527 David Gallegos, 0401
1 MS 9543 Dublin Gonzales, 1498
1 MS 0630 Arthur Hale, 9600
1 MS 0630 Barry Hess, 9610
1 MS 6233 Brian Jones, 1138
1 MS 8962 Philip Kegelmeyer, 9159
1 MS 5634 Brian Kennedy, 0621
1 MS 8944 Joseph Lewis, 9019
1 MS 9537 John Mareda, 0899
1 MS 9543 Patrick Milligan, 1498
1 MS 9329 James Muntz, 0838
1 MS 9151 Leonard Napolitano, 8900
1 MS 5534 Andrew Scholand, 1243
1 MS 8944 Matthew Schrager, 9019
1 MS 9538 Peggy Schroeder, 0805
1 MS 9537 Judith Spomer, 0899
1 MS 9342 Andrew Steele, 0931
1 MS 9549 Andy Tzou, 1482
3 MS 8944 Tracy Walker, 9036
1 MS 8944 Jeffrey West, 0899
1 MS 0899 Technical Library, 9536 (electronic)