Fall 12-18-2014

Integrating Fuzzy Decisioning Models With Relational Database Constructs

Erin-Elizabeth A. Durham
Georgia State University

INTEGRATING FUZZY DECISIONING MODELS

WITH

RELATIONAL DATABASE CONSTRUCTS

by

ERIN-ELIZABETH ANDERSON AUGARD DURHAM

Under the Direction of Robert Harrison, PhD

ABSTRACT

Human learning and classification is a nebulous area in computer science. Classic decisioning problems can be solved given enough time and computational power, but discrete algorithms cannot easily solve fuzzy problems. Fuzzy decisioning can resolve more real-world fuzzy problems, but existing algorithms are often slow, cumbersome and unable to give responses within a reasonable timeframe to anything other than predetermined, small dataset problems. We have developed a database-integrated highly scalable solution to training and using fuzzy decision models on large datasets.

The Fuzzy Decision Tree algorithm is the integration of the Quinlan ID3 decision-tree algorithm together with fuzzy set theory and fuzzy logic. In existing research, when applied to

the microRNA prediction problem, Fuzzy Decision Tree outperformed other machine learning algorithms including Random Forest, C4.5, SVM and Knn.

In this research, we propose that the effectiveness with which large dataset fuzzy decisions can be resolved via the Fuzzy Decision Tree algorithm is significantly improved when using a relational database as the storage unit for the fuzzy ID3 objects, versus traditional storage objects. Furthermore, it is demonstrated that pre-processing certain pieces of the decisioning within the database layer can lead to much swifter membership determinations, especially on Big Data datasets.

The proposed algorithm uses the concepts inherent to databases: separated schemas, indexing, partitioning, pipe-and-filter transformations, preprocessing data, materialized and regular views, etc., to present a model with a potential to learn from itself. Further, this work presents a general application model to re-architect Big Data applications in order to efficiently present decisioned results: lowering the volume of data being handled by the application itself, and significantly decreasing response wait times while allowing the flexibility and permanence of a standard relational SQL database, supplying optimal user satisfaction in today's Data Analytics world. We experimentally demonstrate the effectiveness of our approach.

INDEX WORDS: Database, SQL, Big Data, Query optimization, Fuzzy, Fuzzy ID3, Classification

INTEGRATING FUZZY DECISIONING MODELS

WITH

RELATIONAL DATABASE CONSTRUCTS

by

ERIN-ELIZABETH ANDERSON AUGARD DURHAM

A Dissertation Submitted in Partial Fulfillment of the Requirements for the Degree of

Doctor of Philosophy

in the College of Arts and Sciences

Georgia State University

2014

INTEGRATING FUZZY DECISIONING MODELS

WITH

RELATIONAL DATABASE CONSTRUCTS

by

ERIN-ELIZABETH ANDERSON AUGARD DURHAM

Committee Chair: Robert Harrison

Committee: Rajshekhar Sunderraman

Yanqing Zhang

Irene Weber

Rafal Angryk

Electronic Version Approved:

Office of Graduate Studies

College of Arts and Sciences

Georgia State University

December 2014

DEDICATION

This dissertation is dedicated to my husband Kevin Liam Durham, and to my children, Liam, Isabella, Trenon & Declan, and to my parents Donna Joan Moriarity Anderson and W. Michael Anderson.

Thank you very much for putting up with my late-night, no-sleep, coffee-consuming, unicorn ways. I love you.

And to Christine Monaghan. In honour of Goldfish :).

ACKNOWLEDGEMENTS

I would like to thank my Advisor Dr. Robert W. Harrison, a leading scientist in Machine Learning and Bioinformatics, for providing me the opportunity to work on one of the state of art machine learning techniques "Fuzzy Decision Trees". Without his wonderful support, guidance, and encouragement I could never have achieved such an fantastic accomplishment.

For their advice and support, I am very grateful to my committee: Dr. Raj Sunderraman, the Chairman of the Computer Science Department and a leading scientist in Deductive databases and logic programming, data modeling, and knowledge engineering, Dr. Yanqing Zhang, a leading scientist in Data Mining, Computational Intelligence, Bioinformatics and Machine Learning, Dr. Rafal Angryk, a leading scientist in Big data Analytics, Data mining, and Information retrieval, and Dr. Irene Weber, a leading scientist in many fields including Biology and Bioinformatics, for their complementary scientific expertise, support and guidance throughout this research, as well as for their critical advice on the execution of this dissertation. I am very thankful to the Computer Science Department at the Georgia State University (GSU), the GSU Molecular Basis of Disease Initiative (MBD), and the Georgia Cancer Coalition for supporting this research work.

I would like to expressly thank Dr. Raj Sunderraman for his outstanding support and invaluable advice throughout my academic career. In addition, my gratitude goes out to my colleagues in Dr. Harrison's lab for the useful discussions and rousing debates we've had, most especially Mr. Chinua Umoja, Mr. Michael McDermott, Mr. Andrew Rosen and Mr. Brendan Benshoof. It has been my pleasure. Thank you.

Finally, I would like to present my deep appreciation to my dear family: my husband Kevin and my children, Liam, Isabella, Trenon & Declan for their support and encouragement.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF EQUATIONS

1 INTRODUCTION

Measuring similarity, or membership, between two roughly understood things has long been a holy grail in computer science. One can quantify a thing, one can qualify a thing, but we cannot agree that a computer can give us the same answer that a human would have given us when classifying a subjective "thing" in objective terms. One can look at it as a human being and tell you right away what it should be, but explaining "gut feelings" to a computer program just hasn't been the easiest thing to code. Part of the problem is people. If the same community of people created (and tested) a tool as are using that tool, then they are far more likely to think its findings accurate. People's decisions depend on their upbringing, their culture, their life experiences, and a multitude of other factors [8][29]. Having an entirely different community of people involved in the creation/testing of a tool from its eventual audience is almost a guarantee that the eventual audience will not agree with the classification provided by the tool [8]. In the business software engineering paradigm, this is known as involving the stakeholders, early. But in experimentation, when one is looking for a random sample of volunteers to answer a poll, it really isn't an eligible option to have them involved in the requirements and design phases of your experiment. So you're left with explaining "gut feelings" to a computer again: and trying to guess how a random volunteer would interpret the color blue. How many people would say that Jenny has medium blue eyes, how many would say that she has slightly blue eyes, and how many would say that she has green eyes [8]?

Even temperature is subjective. What characterizes warm to one individual may be perceived as hot to another. What characterizes a reasonably cool Spring Day to a person in Maine may be seen as freezing cold to a person from Hawaii. Human decision making is based on a huge set of variables: the context of the decision, the individual's knowledge-base, even the

individual's personal preferences. The incredible volume and continued growth of data[1] and its associated derived knowledge has led to a gap between the amount of data to be reviewed and the ability of a human individual to 'take it all in' in order to make a considered decision. Human Logic and Decision Making has been studied at length by psychologists, philosophers and computer scientists. These studies not only include how humans make decisions, but also how to determine what is valid data upon which the building blocks to make a decision should be founded. Fallibilism, or the understanding that future knowledge can change the viability of past decisions, is a given in Natural Sciences, but is very hard to express in conventional decisioning algorithms in Computer Science.

The automation of human judgment decision-making is an important field in Computer Science – but the question comes down to what model should be used to make the decisions. Should the considered judgment of the population be considered the defacto "truth" or should individual expert opinions be considered the "final say."

Natural language processing often involves removing uncertainty regarding the meaning of a word. In natural language processing, the point of evaluation is to imitate human judgment with regards to word sense disambiguation in determining the relationship between two words / concepts. Giants such as Google research heavily in this arena and developing algorithms to improve their search results [3]. The human language is ambitious and ambiguous, and word-sense disambiguation algorithms can be evaluated either against a standard taxonomy[4], or compiled crawls of web pages which can be used for corpora creation[3]. The Agirre et al. solution of performing standard linguistic analysis against 8 billion documents culled from a web crawl in August 2008, containing approximately 1.6 Terawords [4], is an excellent solution both for increased precision of measurement, as well as for more likely correlation with current

human judgment of word senses. But for other purposes, this web mining solution can be a detriment. One can write a million inaccurate web pages positing that there are 65 states in the United States of America, but that doesn't make it true.

Understanding the leap between data and a human decision is nebulous. Classic decisioning resolves to a true/false answer. Fuzzy decisioning resolves to a degree of membership. The goal of decisioning is to make predictions based on a dataset, mimicking as closely as possible the decision that would have been made by an individual human expert [4]. Computers can do a great deal of this sort of prediction for known values via if-then rules, as long as there is a source of truth to train against. The trick is to get a good accuracy within a reasonable response time. The problem is that one has these completely quantifiable objective measurements, often with multiple qualifications, but as humans one has a difficulty agreeing on what subjective term can be applied to them [30]. And as the subjective term and the objective term are spelled the same and used in the same context in the English language, i.e.:

- What temperature is it outside? It's 65 degrees outside.
- What temperature is it outside? It's warm out.

We have a disambiguation issue [3][4][29].

1.1 PURPOSE OF THE STUDY

So how does one solve this disambiguation issue for large sets of data? Well, solve is a very strong word. One can approximate a solution though, in the same way that a human being can approximate a solution: by making the best estimate that one can, based on the information that one has at the time of the judgment. So how do we do that? It's a subjective, fuzzy problem. Classic decisioning problems can be solved given enough time and computational power, but discrete algorithms cannot easily solve fuzzy problems. Fuzzy decisioning can

resolve more real-world fuzzy problems, but existing algorithms are often slow, cumbersome and unable to give responses within a reasonable timeframe to anything other than predetermined, small dataset problems. The Fuzzy Decision Tree (FDT) algorithm is the integration of the Quinlan ID3 decision-tree algorithm together with fuzzy set theory and fuzzy logic [6]. In this research, our initial goal was to take the FDT algorithm [6] and map it to a facile relational database, using the objects inherent to a database: separated schemas, indexing, partitioning, pipe-and-filter transformations, preprocessing data, materialized and regular views, etc. These database objects are already optimized for use in a database, and we separated the heavy-processing data-manipulation logic and placed it in the database layer with the data itself, expecting excellent results.

One of our initial goals was to work on expanding the GSU fuzzy decisioning freeware tool, FDT, first developed by Dr. Abu-halaweh [6], and therefore for accessibility and ease of use for future end-users, we chose to use the freeware database MySQL as our database platform. To this end, we maintained that the tool should produce immediate results, but also create a datastore with results that could be independently explored afterwards via adhoc querying; thus allowing the end-users the ability to draw further conclusions from the resultant data, and/or manually prune data as desired from the training sets. The idea of growing a community of end-users who could contribute to an open-source solution was toyed with, but for tracking purposes, the current version is freeware only, for which usage permission must be requested from the Computer Science Department.

We first began exploring the ideas of speeding up and optimizing fuzzy analytics with an eye to including databases purely from a storage perspective. Our discoveries as we continued down the path were in some cases quite surprising! For example, in the freeware MySQL

database, we ran into an unexpected implementation issue that guided our design. In very wide (many-featured) implementations, the database maximum per-table restriction of 4096 columns, and per-query (or view) join restriction of a maximum of 61 tables encouraged us to find alternate ways to physically store our data, rather than exactly how it may appear in a source file or on the screen.

Accordingly, we looked at what alternate methods were available to store the data. EAV structures looked good at first, with their extreme flexibility and rigorously segregated key tables, but having to constantly flatten the data into (effectively) materialized views just to see anything of value would take away from one of the benefits of having the data in a static storage format for the proposed end-users of the freeware: allowing the end-users the ability to build adhoc queries and independently explore the data afterwards via adhoc SQL querying in the database, drawing further conclusions from the resultant data. Also, as can be seen on multiple forums of commercial and community editions of existing software today that utilize EAV structures (Magento, Zoho, etc), if the end-users forget to run/schedule the process to flatten the data, it remains un-updated in the materialized views, resulting in end-user frustration with the tool... and as we cannot control our users' actions, and we do not want users frustrated with our tool from such a simple cause, EAV structures were discarded.

NoSQL, with its current media spotlight and flexible format, offered some food for thought during the initial design phases, but proved too difficult to consistently find potential users who would be capable of pulling data back out of the "database," as again one of our initial goals was to improve on the FDT tool - and a tool is only good if it is USEFUL. Most business data analysts in today's scientific community can use SQL and SQL-based tools. Few can yet use NoSQL easily. NoSQL was discarded.

Normalized relational databases by their very nature always maintain updated values, and therefore would not suffer from the EAV detriment of not being easy to query on their own as a stand-alone useful artifact. And a major goal was to create something not only technologically superior to the existing options available, but also to create something USEFUL. Creating something wonderfully clever that is too complicated for anyone to ever be able to actually use seemed like a terrible waste, so the goal of Ease of Use also made it to our Drawing Board.

The most popular querying language is SQL (standard querying language). Support for SQL is built in to all modern database management systems [19] [20] [21] [22] [23] [24], and business-analyst resources who know how to both understand the business drivers as well as how to use the querying language are relatively easy to hire. Relational databases with referential integrity are the easiest method for current IT business analysts to hit the ground running and dig in for answers to common data-related business context questions.

A normalized, relational MySQL database could provide the (effective) benefits of materialized views when required, and in some cases this is an extremely useful method of shortening otherwise inescapably long processing times. Discovering this implementation nugget while digging further into speed-up methods for integrating fuzzy decisioning models with relational database constructs created the basis for a methodology for the optimization of Relational Database Usage involving Big Data, and later a Model Architecture for Big Data Applications using Relational Databases. Despite the media hype that states that everything and everyone is all unstructured data and a-Twitter, in our research we discovered that according to industry surveys like the 2012 IOUG Big Data Strategies Survey [32], for most companies (90% of respondents) the total sum of all data from all sources is only in the multi-terabyte range, and over three quarters (77%) of the respondents considered structured data (transactional database

data) to be most important to their business. From this information, one can infer that the majority of Big Data analysis currently occurring in private industry is largely dealing with structured datasets sized in the multi-terabyte range. In many cases, the volume of data is hard to handle not because the database engine is incapable of handling it, but because either the data model or the application queries were ill-designed: the developers thought only of how the application would be putting the data INTO the database, and not how the data would be coming back OUT [14]. Effective Big Data applications must handle the retrieval of decisioned results based on stored large datasets efficiently. One effective method of requesting decisioned results, or querying, large datasets is the use of SQL and database management systems such as MySQL. But a perceived problem with using relational databases to store huge datasets is the decisioned result retrieval time [14]. Relational databases are often seen as slow by developers because of poorly written Big Data application queries / decision requests. But the business analysts that work with the data daily to perform ongoing discovery and analysis prefer relational databases because they are easy to understand and easy to query using SQL and tools that the analysts already know how to use.

Unfortunately, with the huge volume and variety of data involved, naive or untuned queries can take staggeringly long amounts of time to return results, rendering the Big Data application useless. But there is a silver lining: the fact remains that the calling application does not need all of the data, all at once. You do not need to hold every dollar bill from your bank account in your hands in order to pay the bill at your favorite coffee shop. You only need to have in your hand the amount required to perform the bill-paying activity. By this same argument, in order to answer many Big Data decisioning questions, all of the data points from the stored Big Data do not need to be returned to the calling application: only enough to answer

the question being asked. The calling application needs a question answered. Using the database as just a receptacle to hold data sets, while the calling application does all of the work, is a misuse of available resources and a waste of bandwidth. Imperative database programming elements including stored procedures, functions and triggers that sit directly in the database and do not incur the overhead of network traffic, and have direct access to the data itself, allow the preponderance of the heavy-lifting data analysis work to be performed at the database layer and are supplied by every major database vendor [19][20][21][22][23][24]. In most cases this is faster than the well-known map-reduce paradigm [31]. That's largely because the basic idea for this framework has existed in parallel SQL database management systems for over 20 years [31][18][12][16][15].

As we discovered, the biggest problem in dealing with large datasets, up to and including Big Data, is not storing the data itself, but is deciding what data is useful for a particular decision request. Different types of data lend themselves to different analysis techniques. Most traditional Business Intelligence employs descriptive analysis techniques to create reports including statistical functions (max, min, mean, etc.) to describe the population of data that is being reviewed. The reports often include summary data tables, graphics (charts), and explanatory text. This sort of quantitative analysis is typically used with structured data. With inductive analysis, you begin by examining concrete examples of your data. You then apply qualitative methods to the data population in order to understand the domain and determine what the important factors are. This works especially well for unstructured data where you may not know what insights to expect from your data population.

The optimization of fuzzy decisioning algorithms in order to approximate expert human judgment and disambiguate classifications is incredibly important for working towards the

ability to efficiently work with very large, if not Big Data, datasets. The purpose of the experiments was to determine whether the proposed algorithm could be implemented to take advantage of a database's procedural elements, in order to facilitate swifter decisioning of larger data sets.

1.2 CHAPTER ORGANIZATION

The experiments performed are outlined in a series of chapters as follows:

Chapter 2 outlines the optimization strategy for large data analytics in Optimization of Relational Database Usage Involving Big Data. The lessons learned from the optimization were implemented in Chapter 3, "FDT 2.0: Improving scalability of the fuzzy decision tree induction tool - integrating database storage." In our experiments, we created FDT 2.0, and captured the base data, the fuzzification model, and the decision information into a relational database, from which future decisions can be extrapolated. Large biological datasets from HIV, microRNAs and sRNAs were used to measure the effectiveness of the new tool.

Chapter 4, "A Novel Approach to Determine Docking Locations Using Fuzzy Logic and Shape Determination," discusses our first pass at a second tool, MPDA, which attempts to predict noncovalent binding of macromolecules with other ligands. Aided by the lessons learned in our earlier toils, we used a relational database to hold shape and atom information, using the information it contained to calculate charge. We also calculate the gradient of potential for each shape to measure binding affinity. With the shape information and the gradient of potential, a fuzzy determination is created to determine a ranking system of the possible docking locations. Chapter 5, "A Novel Approach to Determine Docking Locations using Fuzzy Shape Recognition," discusses the successful results of the experiment.

Chapter 6 reviews the optimization we developed and discusses an overall Model Architecture for Big Data applications, together with experimental results. Chapter 7 contains our Conclusions regarding the research performed.

2 OPTIMIZATION OF RELATIONAL DATABASE USAGE INVOLVING BIG DATA

Effective Big Data applications dynamically handle the retrieval of decisioned results based on stored large datasets efficiently. One effective method of requesting decisioned results, or querying, large datasets is the use of SQL and database management systems such as MySQL. But a problem with using relational databases to store huge datasets is the decisioned result retrieval time, which is often slow largely due to poorly written queries / decision requests. In reviewing the literature surrounding database usage, especially relational database usage, it became increasingly apparent that a myth existed that relational databases were not well-suited to large handling amounts of data. In and of itself, that statement sounded ludicrous. We investigated the root cause of this issue, and determined that the main source of the myth had more to do with the volume of data being passed between the database and the application for processing, than with the database itself. The people developing the Big Data applications were without an understanding of the capabilities of the database itself for performing the processing on the database side, and were either creating extremely inefficient queries or else were requesting back all of the data (instead of only the pertinent data), every time a decision was requested. We present a model to re-architect Big Data applications in order to efficiently present decisioned results: lowering the volume of data being handled by the application itself, and significantly decreasing response wait times while allowing the flexibility and permanence of a standard relational SQL database, supplying optimal user satisfaction in today's Data Analytics world. We experimentally demonstrate the effectiveness of our approach.

2.1 INTRODUCTION

Big Data applications span multiple business contexts, from social media to financial interactions to scientific computing. The incredible volume and continued growth of data [1] and its associated derived knowledge has led to a gap between the amount of data to be reviewed and the ability of a human individual to 'take it all in' in order to make a considered decision. Accordingly, companies have addressed this issue by creating applications to automate the process of decisioning: providing graphs, dashboards and useful subsets of the data to the human users that need the data for their day-to-day jobs.

But what is Big Data? The term is thrown around so much in business and scientific contexts, with such varied meanings [2][3][4][5][6], that a quick moment to clarify the meaning in the context of this paper is not out of place. Big Data is a blanket term for any collection of data sets considered to be too large and complex to easily manage, curate and process using traditional means: i.e. database management tools and/or data processing applications. The main descriptors were defined over 10 years ago as the 3 Vs: Volume, Velocity and Variety [7]. Volume is simple to understand: a lot of data. Big Data sizes are a moving target [8][9], but in most current contexts, it usually includes data sets with sizes from a few terabytes [10][11] to many petabytes [12]. Or more simply put, "Big data is what happened when the cost of storing information became less than the cost of making the decision to throw it away." [13]

Velocity also is fairly self-explanatory: the data accumulates swiftly. Big data is usually composed of collected observations over time, and/or recorded transactional data points, for a distinct set of entities. And this set of entities is usually much smaller than the total number of observations being collected. Most data becomes Big due to repeated entries for the same entities over time and/or space [14]. For example, a retailer might have millions of customers,

thousands of inventory items and hundreds of stores, but will log billions of individual transactions annually. Telecommunications traffic entities can include voip devices, cable modems and subscribers, and are limited to the number of physical devices in use, but will log statistical data on a per-minute to per-hour basis for each entity, accruing millions and millions of records daily. Scientific measurements can produce data that is even bigger: the Large Hadron Collider experiments alone generate more than 20 petabytes per year [12].

So what about the last term, Variety? Variety means that the data comes from a lot of different places. Does that mean that there is structured data from different sources involved, such as different systems, different databases, different applications, etc? Yes, very likely. What kind of data is considered structured data? Data that conforms to an explicitly defined data model is structured data. Though this type of data is often found in a database, this may also be found in predetermined record fields inside a file, or set of files. Does the term "Big Data" mean that there must be semi-structured data involved? No. Can the Variety of data being saved include semi-structured data? Potentially, if the company paying the bills wants to save that kind of data, or wants to use that kind of data in their analysis. What kind of data is considered semi-structured? XML documents, JSON documents, email and EDI are all examples of semi-structured data. (Basically, semi-structured data usually has a defined structure, but it is not data that would normally conform to a relational database structure.) Does Big Data always include unstructured data? No. Can unstructured data be included in the Variety of data being saved? Possibly, if the company paying for the storage of the Big Data and the subsequent analysis of that information wants to save and potentially use that kind of data in their analyses. What kind of data is considered unstructured? Information without a pre-defined organization (structure), and/or without a defined data model that is meaningful to the Content of the data itself, is usually

considered to be unstructured. This often includes the textual content of any kind of documents, texts, tweets, instant messaging chats, presentations, emails, etc., but can also include audio, video and image files.

Most companies largely focus on the analysis of structured data, also known as traditional Business Intelligence [15]. Aside from the well-understood benefits of Business Intelligence that has been around for multiple decades, what do we gain by having more and varied data to work with? The ability to save huge amounts of data in traditional databases has allowed unparalleled amounts of traditional trending, forecasting, and pattern detection. People often are oblivious about just how much information is being collected. For example, stores do indeed keep track of all of their customers' purchases over time: that information belongs to the store, and can be scrutinized for insights into the lives of their customers. The analysis of this data is not a "neutral or unguided process" [16]. A company has to pay for the storage and for the analysis of their data. It is not to be expected that the company would perform these activities without a reason: under normal circumstances, do you often personally drive down a road that does not contain at least one destination that you would like to reach? Driving takes up your time, your money (fuel for the vehicle), and your assets (the car and you): you are most likely to use these things for reasons that will benefit you. A company is made up of individuals: they are no different.

Therefore the largest amounts of money driving the analysis of Big Data will be the same as those that previously drove traditional Business Analysis: sales and marketing [17]. Do you eat out on Wednesdays for lunch? With friends? Do you shop for groceries on the way home from work on Mondays? Do you always buy brand name sodas? Do you only purchase buy-one-get-one-free cereal? Do you tend to buy the same brand of hot dogs, but vary over the type

of hot dog buns, depending on what's on sale? How about your clothing and furniture purchases? Do you tend to purchase from big retailers, or boutique or specialty stores? Are you pregnant? Do you even know about it yet? Depending on where you shop, your store may well know about it before you do [18]. Needless to say, with the unparalleled level of data being stored, privacy is a big concern as well [19][20]. This is especially true in its downstream implications. Not only is it unnerving that a retailer knows that you prefer purple pajamas over any other color, and that you always purchase your holiday gifts the day before Christmas, employers and insurance companies have now made a practice of purchasing this historical customer spending information from these retail companies in order to develop employee profiles [21]. Again, currently it is often about sales and marketing: Blue Cross and Blue Shield of North Carolina stated that their use of the purchased data was to try to "sell you something that can get you healthier" [21]. But scrutinizing this historical customer spending information can give employers other insights that are generally not currently exploited (but could easily be) from the same data set, especially when an employer is researching a new hire: do you consistently buy Late Birthday gifts or cards? Do you consistently purchase alcohol or tobacco products? Or for an existing employee: Are you often shopping on days when you've called in sick? The majority of the insight and advantage from large-scale data analytics will be directed at answers to the questions that the companies have an interest in asking, and these questions are usually tailored to the specific needs of their business.

One of the biggest problems with any kind of traditional Business Intelligence data analysis, is that the data is always filtered. The lowest-level employee working on an assembly line will by definition see more of the day-to-day interaction with the product on the assembly line than the company president. The company president actually has the least granular data

about the product at his fingertips: to make critical operations decisions, he must rely on information that has been filtered through multiple systems and rolled-up, summarized reports.

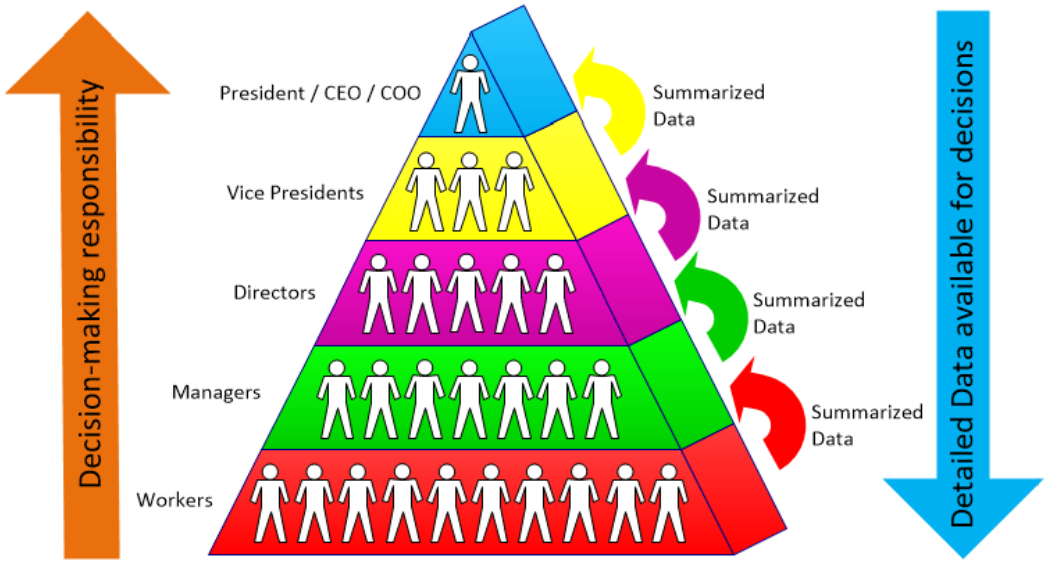


Figure 1 **Traditional Business Intelligence: the people who must make the most impactful decisions tend to have the least granular data.**

The newer concepts in Business Intelligence include the analysis of semi-structured and unstructured data, both from company-held and public sources. These public sources can include anything posted on the web such as personal webpage pages and blogs, to Twitter tweets, Facebook, Instagram, and Google+ posts, YouTube uploads, and Pinterest pins. The textual content of the unstructured data is not a "silver bullet" or cure-all, but analysis of publicly available social media data and can often be used to provide swift customer service responses [17], and direct public relations opportunities [22][23] as well as to develop useful predictive models regarding the opinions or sentiments of online responders (anyone online who feels the urge to tweet or post).

After all, how many times do you praise customer service versus complain about it? It is expected that if you purchase a pizza, it arrives on time. It is expected that if you pay for an item, that it performs exactly as expected. If this doesn't happen, then you complain. The current fastest way to a customer service representative's ear is not to pick up the phone, but rather to tweet to the world about how much the company is disappointing you using their company name as the hashtag [24]. No company wants their online name dragged through the ether-mud, and therefore this online whining [25] will in many cases instantly blackmail the company into quickly (and publicly) remediating the "problem" that you encountered [24]. Most unstructured data mining is currently focused on marketing and IT - getting better information about what current and potential customers are interested in right now, early detection of customer problems with a particular product (or a competitor's product), new marketing stratagems and business practices, price changes, etc. But due to the prevailing public sentiment regarding the freedom to vent online, the general public now voluntarily gives up a great deal of additional information – even categorizing it with hashtags for easier machine searching and automated processing - and sometimes human resources can also gain insight into reasons for employee turnover, or public relations and marketing can take advantage of a power outage during the super bowl to turn a costly advertising loss into a Social Media win [22].

But while mining unstructured data can allow a company to get customer feedback data directly from "the horse's mouth" as the old saying goes, caution and good sense are still required. Using search and indexing technology to monitor publicly-available unstructured data is a useful strategy, but it doesn't always yield usable results [26][27]. An example would be the 2011 big data project that analyzed tweets and other social media outlets to attempt to predict the U.S. unemployment rate via sentiment analysis, by month, based on word counts in these

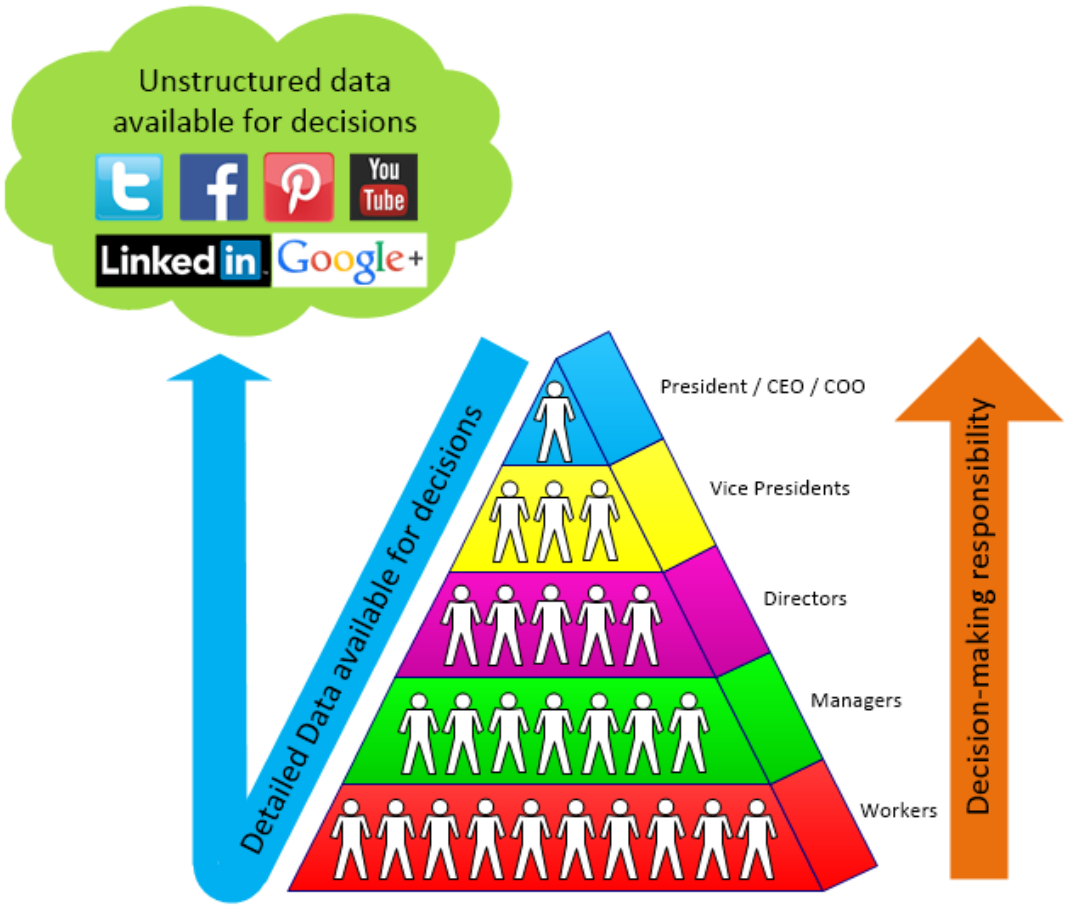


Figure 2 **Analysis of semi-structured and unstructured data can provide direct and detailed information about what current and potential customers are interested in right now.**

unstructured data sources for words such as jobs, unemployment and classifieds [26][27]. In their excitement over finding an upswing in the sentiment analysis word count in October 2011, researchers didn't notice that their word choice was fouled: Steve Jobs had died, and it had lit up the Twitter feeds, regardless of the employment or unemployment of the tweeters. Not only that, but many sources now use your online Friends on Facebook, and LinkedIn to determine your "character and capacity" for loans and employment opportunities [28]. In the USA, credit agencies commonly are required to drop information off an individual's credit report after 7 years (10 years for a bankruptcy proceeding), but information listed on Google, Twitter, and Facebook never dies.

The biggest problem in dealing with Big Data is not storing the data itself, but is deciding what data is useful for a particular decision request. Different types of data lend themselves to different analysis techniques. Most traditional Business Intelligence employs descriptive analysis techniques to create reports including statistical functions (max, min, mean, etc.) to describe the population of data that is being reviewed. The reports often include summary data tables, graphics (charts), and explanatory text. This sort of quantitative analysis is typically used with structured data. With inductive analysis, you begin by examining concrete examples of your data. You then apply qualitative methods to the data population in order to understand the domain and determine what the important factors are. This works especially well for unstructured data where you may not know what insights to expect from your data population.[29]

2.1.1 Data needs to be accessible to be useful

The most popular querying language is SQL (standard querying language). Support for SQL is built in to all modern database management systems, and business-analyst resources who know how to both understand the business drivers as well as how to use the querying language are relatively easy to hire. Referential databases are the easiest method for current IT business analysts to hit the ground running and dig in for answers to common questions such as:

- What happened?
- Why did it happen?
- What will happen next?
- Will it happen again?

But with the huge volume and variety of data involved, naive or untuned queries can take staggeringly long amounts of time to return results, rendering the Big Data application useless.

The biggest issue for Big Data applications is a matter of asking the Right Question. The most famous Big Data question was posed in the fictional work The Hitchhikers Guide to the Galaxy "The answer to the great question of life, the universe and everything." It took seven and a half million years to solve by the computer Deep Thought ...and according to the fictional work, it was finally calculated: forty-two [30]. Which, obviously, is a farcically useless datum. This highlights one of today's most obvious issues with the Big Data environment: the problem isn't putting the data into the database, the problem is getting useful data back out [14].

Previous articles explicitly depicted difficulties of storing all of the data from a very large relational dataset in memory [14] in order to return it to a calling application. But the fact remains that the calling application does not need all of the data. You do not need to hold every dollar bill from your bank account in your hands in order to pay the bill at your favorite coffee shop. You only need to have in your hand the amount required to perform the bill-paying activity. By this same argument, in order to answer many Big Data decisioning questions, all of the data points from the stored Big Data do not need to be returned to the calling application: only enough to answer the question being asked.

The calling application needs a question answered. Using the database as just a receptacle to hold data sets, while the calling application does all of the work, is a misuse of available resources and a waste of bandwidth. Imperative database programming elements including stored procedures, functions and triggers that sit directly in the database and do not incur the overhead of network traffic, and have direct access to the data itself, allow the preponderance of the heavy-lifting data analysis work to be performed at the database layer and are supplied by every major database vendor [31][32][33][34][35][36]. In most cases this is

faster than the well-known map-reduce paradigm [37]. That's largely because the basic idea for this framework has existed in parallel SQL database management systems for over 20 years [37].

2.2 A NEW MODEL

"I checked it very thoroughly," said the computer, "and that quite definitely is the answer. I think the problem, to be quite honest with you, is that you've never actually known what the question is"
-- Deep Thought, The Hitchhikers Guide to the Galaxy[30].

2.2.1 Considerations

The requirements for any given piece of software can be divided into two basic categories: functional and nonfunctional. Functional requirements are descriptions of what the application should do: services provided, how the application should react to inputs, and how it should behave in particular scenarios [38]. They depend strongly on the type of software being developed, the expected users/audience, and the type of environment where the software is expected to be used. Nonfunctional requirements are constraints on the application's services or functions: timing constraints, security restrictions, storage requirements, standards, etc. These often apply to the application as a whole rather than to any one individual features or service in particular [38]. The problem is that nonfunctional requirements may be more critical than functional requirements. If they are not met, then the application may be useless to its expected audience.

2.2.2 Good Software

There are three basic things that every piece of software should be:

1. Usable
2. Dependable

3. Maintainable

Number one in the list relates to the functional requirements, and numbers two and three contain the nonfunctional requirements. But in in examining the reasons why a customer will choose to not use a piece of software, especially a web application, frustration over lag (long wait times between when an application request is sent and when the application responds) far outweighs the benefits of all the bells and whistles. Though this especially comes into play with secured web applications, where slow application responses can have additional cascaded effects [39] up to and including timing out the customer from the application entirely, it is an effective nonfunctional requirement in the building of the FDT-Database as well. Just because a tool is processing a ton of data, does not mean that the end-user will give any consideration to the volume and breadth of the decisioning requirements. The extreme frustration generated from the inability of the software to meet the nonfunctional requirement of a timing constraint can engender such a bad feeling towards an application (almost a hatred) that customers will unreservedly lambaste it for taking even a few seconds longer to respond than expected: a quick google search of the words "I hate lag" yields around 14 million results, for different software, websites and devices. This is not a small concern for a human-facing application.

2.2.3 Back to Basics: Engineering Principles

"Other Engineering disciplines have principles based on the laws of physics, biology, chemistry or mathematics... Because the product of software engineering is not physical, physical laws do not form a suitable foundation. Instead, software engineering has had to evolve its principles based solely on observations of thousands of projects."
-- Alan Davis, IEEE, November 1994 [40]

Software principles have been derived from the collected wisdom people who learned through experience. They will continue to evolve as the discipline grows. Experience is gained by practice: as more experience is gained, the collected wisdom grows and new principles evolve from the massed experience.

Davis's principles are a tried and true checklist of back-to-basics items that any application refactoring effort should make the effort to walk through. The first item to address is Davis's principle number 10, "Get it Right before you make it Faster [40]." Though of course increased speed is always a goal, it was a mandatory functional requirement that the cable modem port data analytics return accurate decisioned results. Davis's principle number five, "Evaluate Design Alternatives [40]," discusses the need to examine a variety of application architectures and algorithms after the requirements have been agreed upon. That was definitely something that that needed to be examined. Davis's principle number seven, "Use Different languages for Different phases [40]," is incredibly clear about pointing out that using object orientation for all phases, when it is not what is optimal for a particular phase, is useless. There's an old saying that states, 'When all you have is a Hammer, then every Problem looks like a Nail.' The problem lies in that the drive to find simple solutions to complex problems, which often blinds developers to solutions outside their immediate comfort zone, forcing draconian non-functional requirements such as always using ORM, or only using Java (or any other single language) as part of the development standards, etc., instead of determining where each portion of the solution would be optimally built. Making everything use the same language does not make the solution simpler. In many cases, it can make a solution much harder. What is simple to accomplish in one language may be incredibly difficult to accomplish in another.

In the original version of the case study software, a Big Data web application to interpret cable modem port analytics, the application layer encapsulated all of the business logic. The Big Data application used a naive inline SQL query to the database, which would then send all of the requested data back to the application, which would then manipulate the data to provide a decisioned result: essentially doing the processing work in two places. This process was expensive and painful, as the I/O to repeatedly export the full data set from the database and then perform decisioning in the web application layer was limited by both the processor speed and the cache read/write speed (not to mention the network speed itself, depending on the end-user's location) – none of which can be controlled by the application developers. This strategy was difficult to maintain, as any changes to the business logic would require a change to the application layer; and at runtime the complex, weighty queries were sent to the database for processing by end-users on-demand, sometimes causing lengthy waits as the Big Data application waited for its results to be returned from the database for further processing, especially with the larger datasets. Here also was an area that needed to be examined.

2.2.4 Breaking it Down: Examining The Alternatives

More than 20 years ago, support for procedural elements was wrapped into the SQL Standards [41]. In 1996, procedural elements, views, and metadata elements were officially added to SQL (as an extension of SQL-92)[42]. As perspective, Sun released the first public implementation of Java in 1995 [43]. In 1999, procedural elements were so far incorporated into standard DBMS implementations, that database vendors were touting their merits at conferences worldwide [44][45]. All major database vendors support stored procedures [31][32][10][34][35][36].

So what are these procedural elements? Stored routines (procedures, functions and triggers) are chunks of code stored directly in the database. They can use standard coding constructs such as IF, CASE, LOOP, WHILE, etc., and they can accept and return parameters, including returning result sets. SQL is an imperative language, i.e. it is inherently canted towards a procedural programming paradigm; and unlike the declarative model of Object-Oriented languages, a procedural programming paradigm allows the programmer to specify an exact step-by-step process to solve a particular need (or problem). These procedural elements in SQL are especially useful for culling extensive, expensive, and/or complex processing of data out of the application layer; and simplifying application logic by then allowing the application to call a single stored procedure instead of multiple inline queries. Procedural elements, stored within the database itself, avoid the overhead of network traffic altogether as they are stored within the database engine and therefore have direct access to the data to be manipulated. As the syntax is already checked when the stored routine is compiled into the database, there is a lesser likelihood of data corruption via application misusage of dynamically generated inline SQL queries. With stored procedures acting as an API, the application does not have to know the details of the relational database implementation, smoothing over the Object-Oriented application complaint of Object-Relational Impedance Mismatch. Again going back to Davis's rule "Use Different Languages for Different phases [40]," by breaking out the data-manipulation portion of the algorithm, it is no longer necessary to force a nonfunctional requirement of object-oriented design for all layers of the application. Davis's principle number 23, "Encapsulate [40]," also lends itself to this approach. As stated in his article, it is a simple concept that results in software that is easier to test, use and maintain.

2.3 EXPERIMENT

2.3.1 Solution Viability

The purpose of the experiments was to determine whether the proposed algorithm could be implemented to take advantage of a database procedural elements, in order to facilitate decisioning larger data sets. Three (3) iterations of the request for a decisioned result set were used for the purpose of the experiment. The first was serviced by a generic, straightforward database SQL query submitted by the development team; the second was serviced by an optimized version of the same generic, straight-forward database SQL query; and third was serviced by the proposed model. Comparisons were created using the volume of data required to complete the database transaction, the time required to run the database transaction, and the number of records returned to the calling application.

2.3.2 Datasets

In a Big Data environment, the largest problem is not obtaining the data or even storing it, but rather getting useful answers derived from the raw data within a reasonable timeframe. One application requiring said swift, useful answers is deriving port analytics for the modems on a cable modem termination system (CMTS). A CMTS is the piece of equipment that a cable company uses to connect each of their headends to the Internet on one side, and their subscribers on the other. In order to provide continuous high-speed data services to its subscribers, a cable company needs to monitor not only the reported health of the CMTS itself, but also the health of each modem in a subscriber location. The values of multiple data points can be used to encapsulate the health of a typical modem, including signal noise ratio, T3 and T4 timeouts and

code word errors [46]. These values can be further rolled up to give a measure of the health of the ports on the CMTS itself.

For the case study experiment, to create their Big Data application, a survey was conducted to determine what database management system (DBMS) would provide the best performance and maintainability for the size of data (around 10 TB for a rolling 6 month period) to be preserved, and alternatives (including both SQL and NoSQL) were examined. MongoDB, specifically, was evaluated closely over several months, but was both unable to keep up with the write and throughput load, and was very difficult for end-users to manipulate afterwards in order to analyze the captured data (getting useful answers). The relational DBMS, Oracle MySQL, was eventually chosen for throughput speed and ease of use.

Internal industry experts had defined the evaluation criteria for each of the 9 attributes to be evaluated: ranges were defined for Red/Bad, Yellow/Warning, and Green/Good. Raw data was collected for each modem on a periodic basis between 15 and 60 minutes. The raw data was partitioned into daily segments, over multiple relational tables.

To make useful decisions about the collected data, it was required that a current measurements snapshot be created against the evaluation criteria. The twist was that each attribute had to be evaluated on a basis of net difference between the most recently collected data and the next most-recent valid set of data collected at least 60 minutes previously.

2.4 RESULTS

2.4.1 Iteration 1: Naive, inline query

Given a defined set of ranges, a straightforward method to calculate the set of records to be displayed would be to take the set full set of records for the current collection and join them

with the full set of records from the previous collection, determining the net difference between the attributes of the two sets in order to perform classification range evaluation (Red, Yellow, Green).

```
for all records in set
for each attribute to be graded
     case (previous – current)
          if value in range_1 then RED
          if value in range_2 then YELLOW
          if value in range_3 then GREEN
```

When the database was initially created, this naïve method gave a reasonable response time (<1s), but as the volume of collected data grew to its current size, the naïve method understandably took longer and longer to obtain results. Using this naïve method as an inline query in the Big Data application, approximately 187G of raw data would be processed to obtain a calculation each time an individual report was run. At the peak volume record set during the testing period, the naïve method averaged 26.5 minutes to process through the approximately 300 million raw relational records gathered hourly into a <100,000 record "snapshot" evaluation response.

2.4.2 Iteration 2: Optimized, inline query

Tuning the naïve query to prune down the decision tree for the result set did help significantly, lowering the response time by more than a factor of ten, but for larger customers this still meant that each individual report did not return data for over 2 minutes – an unacceptably high wait time (lag). Clearly, a different approach was required.

2.4.3 Iteration 3: Proposed model

As suggested in section II.D, it was proposed that in order to optimize the query response time for customers, that the complex queries be culled out of the application layer, and constructed via a stored procedure into an indexable temporary relation / materialized view. Refreshed periodically, faster and simpler queries could then be drawn out of the materialized aggregate view to populate the graphical reports for calling Big Data application.

2.4.3.1 Algorithm:

1. Create a shell structure that encompasses all required columns, including
    a. current values
    b. previous values
    c. net values
    d. other identifying information for required joins
2. Insert the set of devices that meet the business requirements
3. Retrieve the current data points from raw data and update the appropriate fields
4. Retrieve the previous data points from raw data and update the appropriate fields
5. Calculate the net differences and update the appropriate fields
6. Apply the business logic ranges (using the net differences) to evaluate each record

So that there would never be a situation where the cable companies did not receive “live data” on their graphical snapshot reports, the data from the previous evaluation would remain (prominently displaying its timestamp) available for review until the latest data calculations were fully complete.

2.4.4 Performance Analysis:

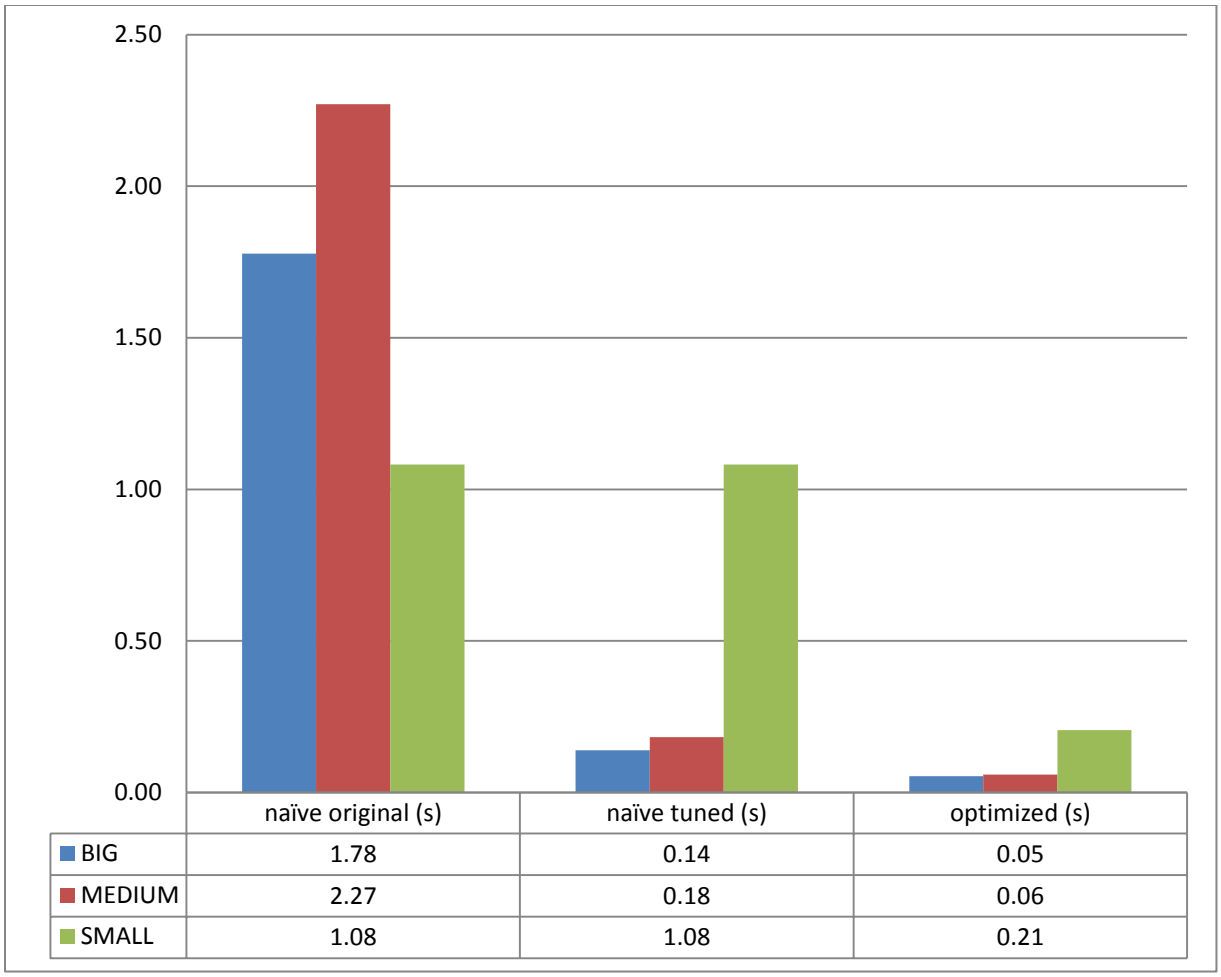
As the previous and current raw data to be evaluated for the net comparisons is now selected in different steps, this process immediately halves the amount of data being processed in memory at any one time as compared to the naïve algorithm.

Table 1 Data set characteristics

	SMALL	MEDIUM	LARGE
number of modems for which data was collected(K)	0.5	62.4	895.9
naïve query (raw file size in G)	0.3	28.1	187.3
optimized query (raw file size in G)	0.1	14.0	93.6

The larger the raw data became, the more evident it became that the materialized view optimization process was much more efficient. The application was able to obtain the required swift, useful answers regarding cable modem port data analytics from the millions of records of raw data within a reasonable timeframe using the outlined method. By making use of the indexable temporary relation / materialized view, the query-retrieval time was cut down 33x. This technique is especially useful in a scenario when calculations can be reused over a period (in the case study, 15-60 minutes), as the results are cached in the indexable materialized view and are thus available for swift retrieval without necessitating the overhead of recalculation.

Dividing the total query processing time by the number of modems, one can see that while a benefit is achieved by using the algorithm even in cases of smaller datasets, the largest benefit is felt when addressing bigger datasets



	naïve original (s)	naïve tuned (s)	optimized (s)
BIG	1.78	0.14	0.05
MEDIUM	2.27	0.18	0.06
SMALL	1.08	1.08	0.21

Figure 3 Query cost per datum performance comparison.

2.5 CONCLUSIONS AND FUTURE WORK

Big Data applications cover multiple contexts, providing valuable business insights into very large data sets. Big Data usually is defined to mean data sets with sizes beyond the ability of commonly used software tools to capture, curate, manage, and process within a tolerable elapsed time. Big data sizes are a constantly moving target: as of 2012 these sizes range from hundreds of gigabytes to a few dozen terabytes to several petabytes in a single data set. The largest Big Data datasets can range to many petabytes in a single dataset, but according to the

2012 IOUG Big Data Strategies Survey sponsored by Oracle [47] almost 90% of respondents reported that that the total amount of data managed (all sources) was under 1 petabyte. Of that data, over three quarters (77%) of the respondents considered structured data (transactional database data) to be most important to their business.

From this information, one can infer that the majority of Big Data analysis currently occurring in private industry is largely dealing with structured datasets sized in the multi-terabyte range. In many cases, the volume of data is hard to handle not because the database engine is incapable of handling it, but because either the data model or the application queries were ill-designed: the developers thought only of how the application would be putting the data INTO the database, and not how the data would be coming back OUT [14].

This work presented a model to re-architect Big Data applications in order to efficiently present decisioned results: lowering the volume of data being handled by the application itself, and significantly decreasing response wait times while allowing the flexibility and permanence of a standard relational SQL database, supplying optimal user satisfaction in today's Data Analytics world.

Effective Big Data applications must handle the retrieval of decisioned results based on stored large datasets efficiently. One effective method of requesting decisioned results, or querying, large datasets is the use of SQL and database management systems such as MySQL. But a perceived problem with using relational databases to store huge datasets is the decisioned result retrieval time. Relational databases are often seen as slow by developers because of poorly written Big Data application queries / decision requests. But the business analysts that work with the data daily to perform ongoing discovery and analysis like relational databases because they

are easy to understand and easy to query using SQL and tools that the analysts already know how to use.

Using Davis's software engineering principles, we developed a strong model to re-architect Big Data applications in order to efficiently present decisioned results. By taking advantage of the procedural elements in SQL and culling extensive, expensive, and complex data processing out of the application layer, we simplify the application logic by the Big Data application to call a single stored procedure instead of multiple inline queries. Stored within the database itself, these procedural elements not only avoid the overhead of network traffic altogether, but also have direct access to the data to be manipulated. With stored procedures acting as an API, the application does not have to know the details of the relational database implementation: the syntax already having been checked when the stored routine is compiled into the database, there is a lesser likelihood of data corruption via application misusage of dynamically generated inline SQL queries.

We experimentally demonstrated the effectiveness of our approach using a 10TB cable modem port data analytics dataset. The Big Data application in the case study (cable modem port data analytics) is still under development. As the outlined Big Data application model already works comfortably with terabyte-sized data sets, future work will include optimized aggregate reporting and trending over time.

2.6 REFERENCES

[1] Gantz, J., Reinsel, D. "Extracting Value from Chaos," The 2011 IDC Digital Universe study sponsored by EMC, June 2011.

[2] Rouse, Margaret. "What Is Big Data?" Big Data and Cloud Business Intelligence. TechTarget, 30 June 2014. Web. 10 July 2014.

[3] Big Data / Analytics / Strategy / FP&A / S&OP / Strategic Planning / Business Analytics / Innovation. "What is the difference between Busiiness Intelligence and Big Data." LinkedIn [Group page]. Web. 31 January 2013. Retrieved July 01. 2014, from http://www.linkedin.com/groups/What-is-difference-between-Busiiness-1814785.S.210009923

[4] Davis, Jim. "What Kind of Big Data Problem Do You Have?" Web log post. What Kind of Big Data Problem Do You Have? SAS Institute, Inc., 08 Oct. 2012. Web. 02 Mar. 2014.

[5] Payandeh, Fari. "BI vs. Big Data vs. Data Analytics By Example." Foreground Analytics Big Data Studio. N.p., 24 Aug. 2013. Web. 04 Mar. 2014.

[6] "Definitions of Big Data." Definitions of Big Data | Opentracker - Digital Analytics. Opentracker, n.d. Web. 01 July 2014.

[7] Laney, Douglas. "3D Data Management: Controlling Data Volume, Velocity and Variety". Gartner. Retrieved 6 February 2001.

[8] Johnston, Leslie. "Data Is the New Black." Web log post. The Signal: Digital Preservation. The Library of Congress, 14 Oct. 2011. Web. 04 June 2014.

[9] "Community Cleverness Required." Editorial. Nature 4 Sept. 2008: n. pag. Community Cleverness Required. Nature Publishing Group, 3 Aug. 2008. Web. 06 June 2014.

[10] IBM. New IBM Big Data Technology for Dramatically Faster Data Analysis and Decision-Making Enters the Market. New IBM Big Data Technology Enables Faster Data Analysis. IBM News Room, 26 June 2013. Web. 04 Mar. 2014.

[11] "Utilizing Better and Faster Intelligence to Improve Strategic Decision Making: Companhia De Seguros Tranquilidade." Pivotal Software. Pivotal, Jan. 2014. Web. 04 Mar. 2014.

[12] CERN. N.p.: CERN, n.d. Accelerating Science and Innovation - CERN Document Server. CERN, 30 May 2013. Web. 01 July 2014. <http://cds.cern.ch/record/1551933/files/Strategy_Report_LR.pdf>.

[13] O'Reilly, Tim (timoreilly). "'Big data is what happened when the cost of storing information became less than the cost of making the decision to throw it away.' - George Dyson #longnow" 19 March 2013, 8:51PM.

[14] Jacobs, A. "The Pathologies of Big Data," ACM Queue: Tomorrow's Computing Today. vol. 7, issue 6, pp. 1-10, July 2009.

[15] Thurai, Andy, and Basu, Atanu. "Prescriptive Analytics: An Adaptive Crystal Ball." Tech News and Analysis. Gigaom, 22 Mar. 2014. Web. 01 July 2014.

[16] Stanley, Jay. "Big Data and Big Money." Web log post. Big Data and Big Money | American Civil Liberties Union. American Civil Liberties Union, 10 July 2014. Web.

[17] Bhatia, Akash. "Social Business The Rise of Unstructured Data." Infosys - Social Media Business Impact | Structured and Unstructured Data. Infosys Limited, May 2011. Web. 01 July 2014.

[18] Duhigg, Charles. "How Companies Learn Your Secrets." The New York Times. The New York Times, 18 Feb. 2012. Web. 01 July 2014.

[19] Terdiman, Danel. "At CNET's SXSW 'big Data' Panel, Sparks Fly over Privacy." At CNET's SXSW 'big Data' Panel, Sparks Fly over Privacy - CNET. CNET, 12 Mar. 2012. Web. 01 July 2014.

[20] Stanley, Jay. "Eight Problems With "Big Data"" American Civil Liberties Union. American Civil Liberties Union, 25 Apr. 2012. Web. 01 July 2014.

[21] Wieczner, Jen. "How the Insurer Knows You Just Stocked Up on Ice Cream and Beer." The Wall Street Journal. Dow Jones & Company, 25 Feb. 2013. Web. 01 July 2014.

[22] Fung, Katherine. "Oreo's Super Bowl Tweet: 'You Can Still Dunk In The Dark'" The Huffington Post. TheHuffingtonPost.com, 04 Feb. 2013. Web. 04 June 2014.

[23] Watercutter, Angela. "How Oreo Won the Marketing Super Bowl With a Timely Blackout Ad on Twitter | Underwire | WIRED." Wired.com. Conde Nast Digital, 02 Feb. 2013. Web. 01 July 2014.

[24] Palmquist, Matt. "How to Order a Social Media Pizza with Extra Text Mining." Web log post. Strategy+business. PwC Strategy& Inc., 25 July 2013. Web. 01 July 2014.

[25] Karen Merrell Puntney (Karen Merrell Puntney). "except when it is for later posting on the restaurant's FB page as a complaint (seeeeeeeee how horrible!) to get free stuff" 14 July 2014, 1:02PM. https://www.facebook.com/ifyoucantaffordtotip/posts/10152209242393091?reply_comment_id=10152210123308091&total_comments=1

[26] Arellano, Nestor E. "Why Big Data Is Not Always Good Data." IT World Canada. IT World Canada, 8 Sept. 2013. Web. 01 July 2014.

[27] Tucci, Linda. "Big Data Can Mean Bad Analytics, Says Harvard Professor." Big Data Can Mean Bad Analytics, Says Harvard Professor. TechTarget, July 2013. Web. 01 July 2014.

[28] "Stat Oil." The Economist [London, England] 9 Feb. 2013. The Economist. The Economist Newspaper Limited, 09 Feb. 2013. Web. 01 July 2014.

[29] "Big Data car Low-Density Data? La Faible Densité en Information comme Facteur Discriminant." Lesechos.fr. Les Echos News, 03 Apr. 2013. Web. 01 July 2014.

[30] Adams, D. (1979) The Hitchhikers Guide to the Galaxy. Germany: Pan Books.

[31] PostgreSQL: Documentation: 9.3: Procedural Languages. PostgreSQL: The world's most advanced open source database. Retrieved March 1, 2014, from http://www.postgresql.org/docs/current/interactive/xplang.html

[32] MySQL 5.7 FAQ: Stored Procedures and Functions. MySQL Reference Manual. Retrieved March 2, 2014, from http://dev.mysql.com/doc/refman/5.5/en/faqs-stored-procs.html

[33] CALL statement. IBM iSeries Information Center, Version 5 Release 3. Retrieved March 2, 2014, from http://publib.boulder.ibm.com/infocenter/iseries/v5r3/index.jsp?topic=/db2/rbafzmstcallstmt.htm

[34] "Stored procedure." Sysbase Wiki. Retrieved March 1, 2014, from http://www.petersap.nl/SybaseWiki/index.php/Stored_procedure

[35] "Stored Procedures (Database Engine)." Microsoft Technet: SQL Server. Retrieved March 1, 2014, from http://technet.microsoft.com/en-us/library/ms190782.aspx

[36] Stored Procedures and Functions. Oracle Documentation. Retrieved March 1, 2014, from http://docs.oracle.com/cd/B28359_01/server.111/b28318/data_access.htm#CNCPT1776

[37] Pavlo, Andrew, Erik Paulson, Alexander Rasin, Daniel J. Abadi, David J. DeWitt, Samuel Madden, and Michael Stonebraker. "A Comparison of Approaches to Large-scale

Data Analysis." Proceedings of the 2009 ACM SIGMOD International Conference on Management of Data (SIGMOD '09) (2009): 165-78.

[38] Sommerville, I. (2011) Software Engineering, 9th Edition. Boston, Massachusetts: Pearson Education, Inc.

[39] United States Department of Health and Human Services (2013, Oct 20). Doing Better: Making Improvements to HealthCare.gov. [Web log post]. Retrieved Mar 02, 2014, from http://www.hhs.gov/digitalstrategy/blog/2013/10/making-healthcare-gov-better.html.

[40] Davis, A. "Fifteen Principles of Software Engineering." IEEE Software 11.6 (1994): 94+.

[41] ISO/IEC 9075:1992 - Information technology -- Database languages -- SQL. ISO - International Organization for Standardization. Retrieved March 2, 2014, from http://www.iso.org/iso/iso_catalogue/catalogue_ics/catalogue_detail_ics.htm?csnumber=16663

[42] Eisenberg, A. (1996). "New standard for stored procedures in SQL". ACM SIGMOD Record 25 (4): 81–88.

[43] "The History of Java Technology". Oracle. Retrieved Mar 02, 2014, from http://www.oracle.com/technetwork/java/javase/overview/javahistory-index-198355.html.

[44] "Guide to SQL Programming: SQL:1999 and Oracle Rdb V7.1." Oracle. Retrieved Mar 02, 2014, from http://www.oracle.com/technetwork/database/database-technologies/rdb/0307-sql1999-130211.pdf

[45] SQL99, SQL/MM, and SQLJ: An Overview of the SQL Standards. Databases and Information Systems - Institute of Computer Science - University of Innsbruck.

Retrieved Mar 02, 2014, from http://dbis-informatik.uibk.ac.at/files/ext/lehre/ss11/vo-ndbm/lit/ORel-SQL1999-IBM-Nelson-Mattos.pdf

[46] Data Over Cable Service Interface Specifications. Cable Labs. Retrieved Jan 02, 2014, from http://www.cablelabs.com/specifications/CM-SP-PHYv3.0-I08-090121.pdf

[47] McKendrick, Joseph. Big Data, Big Challenges, Big Opportunities: 2012 IOUG Big Data Strategies Survey. Rep. Oracle, Inc, Sept. 2012. Web. 01 July 2014.

3 IMPROVING SCALABILITY OF THE FUZZY DECISION TREE INDUCTION TOOL

Integrating Database Storage

Effective machine-learning handles large datasets efficiently. One key feature of handling large data is the use of databases such as MySQL. The freeware fuzzy decision tree induction tool, FDT, is a scalable supervised-classification software tool implementing fuzzy decision trees. It is based on an optimized fuzzy ID3 (FID3) algorithm. FDT 2.0 improves upon FDT 1.0 by bridging the gap between data science and data engineering: it combines a robust decisioning tool with data retention for future decisions, so that the tool does not need to be recalibrated from scratch every time a new decision is required. In this paper we briefly review the analytical capabilities of the freeware FDT tool and its major features and functionalities; examples of large biological datasets from HIV, microRNAs and sRNAs are included. This work shows how to integrate fuzzy decision algorithms with modern database technology. In addition, we show that integrating the fuzzy decision tree induction tool with database storage allows for optimal user satisfaction in today's Data Analytics world by automatically building a persistent data store of objects that the user can continue to mine using the industry-standard querying language, SQL.

3.1 INTRODUCTION

Decisioning, or the machine emulation of human learning and classification, is a nebulous area in computer science. Classic decisioning problems can be solved given enough time and computational power, but discrete algorithms cannot easily solve fuzzy problems.

Fuzzy decisioning can resolve more real-world fuzzy problems, but existing algorithms are often slow, cumbersome and unable to give responses within a reasonable timeframe to anything other than predetermined, smaller dataset problems. As the volume of data available for analysis grows in the modern world, it is becoming more and more imperative that effective machine-learning solutions are examined that can efficiently handle large datasets.

The effectiveness with which fuzzy decisions can be resolved via the Fuzzy Decision Tree algorithm is significantly improved when using a database as the storage unit for the fuzzy ID3 objects, versus standard flat files and in-memory java objects. Furthermore, we demonstrate that pre-processing certain portions of the decisioning within the database layer can lead to better membership classifications, especially on large datasets. Large biological datasets from HIV, microRNAs and sRNAs were used to measure the effectiveness of the tool. microRNA and sRNA are sequence-based function prediction for RNA in eukaryotes and prokaryotes, respectively. The HIV data are prediction of drug resistance from protein sequence data. Additional datasets from the UCI Repository of Machine Learning Databases and from private industry were used to demonstrate the range of the new tool, from small (<150) to truly large (>400000).

The freeware fuzzy decision tree induction tool, FDT, uses an improved fuzzy ID3 (FID3) algorithm to perform its fast decisioning [1]. Fuzzy sets were coupled with the Quinlan ID3 partitioning algorithm to generate the decision trees that are the basis of the tool's logic [2]. As decision trees are notoriously sensitive to small changes in training data [3], and largely unable to cope well with uncertain/variable data, FDT 1.0's implementation of fuzzy sets and fuzzy reasoning used approximation to deal with the data set noise: uncertainty/inexact data and

fluctuations in data precision, etc. The result was a to create a rigorous and effective decisioning tool [1].

FDT 1.0 is a java-based application which does not account for data retention for future decisions, and therefore needs to be recalibrated from scratch every time a decision is required. It outperforms C4.5 and the genetic algorithm tree on every dataset against which it was tested, and it outperforms Random forest on many [1].

FDT 2.0 captures the base data, the fuzzification model, and the decision information into a relational database, from which future decisions can be extrapolated.

FDT 2.0 brings a comparable accuracy level, with the added benefits of having the training/test sets maintained in a stand-alone database, each dataset now with its own identifiable set of database objects that can be dropped and reused (or maintained, as desired by the tool audience) independently, allowing multiple training/test sets to coexist in the tool without interfering with each other. Additionally, the tool users can independently explore the data afterwards via adhoc SQL querying in the resultant relational database, drawing further conclusions from the resultant data, and/or manually pruning data as desired from the training sets.

The rest of the paper is organized as follows: Section 2 introduces FDT 2.0: a database storage version of the fuzzy decision tree induction tool, including an overview of the new FDT 2.0 algorithm. Section 3 presents our experiments results and Section 4 concludes the paper.

3.2 FDT 2.0: NEW AND IMPROVED!

3.2.1 FDT Algorithm

The FDT algorithm couples the Quinlan Iterative Dichotomiser 3 algorithm to recursively create decision trees, with a fuzzy data / fuzzy membership representation to deal with uncertainty, noise, and outlier data elements that normally would cause the Quinlan ID3 algorithm to falter due to its sensitivity to small changes in training data [1].

There are 4 steps involved in the FDT fuzzy decision tree induction algorithm [1]:

1. Data Fuzzification.
2. Generating the fuzzy decision tree.
3. Converting the fuzzy decision tree into a set of fuzzy rules.
4. Inference.

Step 1 of the FDT fuzzy decision tree induction algorithm involves calculating the membership values of the supplied data, either using a fuzzy membership function supplied by the Domain Experts associated with the supplied data or automatically generated based on the contents of the data itself. Next, is the actual building of the fuzzy decision tree. The training data is recursively partitioned based on the values of an attribute chosen via an information theory measure. Multiple choices exist for the information theories measures and fuzzy membership functions. The fuzzy decision tree (FDT) is then boiled down into fuzzy rules of the form “if p then q.” The final step is inferring matches from the dataset to be tested, against the generated fuzzy rules.

FDT 1.0 is a java-based application that implements the Fuzzy Decision Tree (FDT) algorithm: the integration of the Quinlan ID3 decision-tree algorithm together with fuzzy set theory and fuzzy logic[1]. In existing research, the Fuzzy Decision Tree produced comparable

results and/or outperformed other machine learning algorithms including Random Forest, C4.5, SVM and Knn [1], and is therefore a prime candidate for integration with a database to facilitate larger data set analysis.

3.2.2 FDT 2.0

To create FDT 2.0, we took the FDT algorithm and mapped it to relational database constructs, using the objects inherent to a database: separated schemas, indexing, partitioning, pipe-and-filter transformations, preprocessing data, materialized and regular views, etc. These database objects are already optimized for use in a database, and one separated the heavy-processing data-manipulation logic and placed it in the database layer with the data itself, with excellent results. Using the freeware MySQL 5.5 as the database software, the FDT-Database software performed very well on larger datasets, running into hundreds of thousands of records in the training and test data files.

Each training/test set now had its own identifiable set of database objects that could be dropped and reused (or maintained, as desired by the tool audience) independently, allowing multiple training/test sets to coexist in the tool without interfering with each other. Comparable accuracy results were achieved, and larger datasets could now be classified by the tool. As the training data is held in a database, it is not necessary to rerun the training steps whenever the application is invoked, as it is with the FDT 1.0 too1 [1]. There is no imposed limit to the number of test / train sets that can be stored, and accurate rulings can be added back into the training set (and training steps (b)-(d) can be re-run), to grow the “knowledge-base” of the training set.

3.2.3 FDT 2.0 Algorithm

The biggest difference between the initial FDT algorithm and the algorithm in FDT 2.0, aside from the implementation difference of handling the data in a database instead of via flat files, is that in FDT 2.0 the training data is available to assist in the classification, which can lead to improved accuracy.

3.2.3.1 Training/Learning Steps:

a. Load the multi-variant training data into the training table.

b. Run statistical analysis on the training data to determine the frequency, probability, entropy, dominant value, and other statistical measures for each attribute and category.

c. Based on the training data, determine the if-then decision tree.

d. Based on the training data, determine the fuzzy set membership qualifications.

3.2.3.2 Classification/Testing Steps:

a. Load the multi-variant test data into the test table.

b. Predict the classification for the multi-variant test data.

   i. Determine memberships

   ii. Evaluate fuzzy rules

   iii. Combine outputs of fuzzy rules

   iv. Determine crisp classification prediction

c. Measure the accuracy of the class prediction.

d. (optional) Add accurate classification rulings back into training table, and rerun training.

3.3 EXPERIMENT

3.3.1 Solution Viability

The purpose of the experiments was to determine whether the Fuzzy Decision Tree algorithm could be implemented to take advantage of a database storage structure, in order to facilitate decisioning larger data sets. Initial comparisons were run using the testing databases from UCI Repository of Machine Learning Databases to make comparisons against the earlier implementations of the FDT that had not used a database storage structure. Large biological datasets from HIV, microRNAs and sRNAs were used to measure the effectiveness of the tool. Later experiments utilized classified large data sets from private industry.

3.3.2 Datasets

For the first set of experiments with the FDT classification tool, we used four (4) of the publically available datasets from the UCI machine learning repository [4]: Shuttle, microRNA, sRNA and Iris. This included the biological datasets for sequence-based function prediction for RNA in eukaryotes and prokaryotes, microRNA and sRNA, respectively. The datasets varied widely in type and attributes, from Shuttle with a set of 58000 multivariate items with 9 integer attributes and 7 possible classifications; to Iris, with a set of 150 multivariate items with 4 real attributes and 3 possible classifications. The data sets were pre-processed into SVM-like training and test files. Then the datasets were run through the FDT classification tool, and the prediction results evaluated.

After completing these experiments satisfactorily, we went looking for larger data sets to classify. For the larger datasets, we used classified datasets of HIV-1 protease mutant

Table 2 Properties Of The Datasets Used During Experiments.

Name	Source	Records	Features	Categories	Size
Pharos-QoS	Momentum Telecom	409802	9	4	XL
TPV-HIV	Stanford University	10228	210	2	L
Shuttle	UCI machine learning repository	58000	9	7	L
microRNA	UCI machine learning repository	4425	8	2	M
sRNA	UCI machine learning repository	1812	8	2	M
Iris	UCI machine learning repository	150	4	3	S

structure/inhibitor complexes from the genotype-phenotype datasets at Stanford University [5]. The HIV data are classified prediction data of drug resistance from protein sequence data.

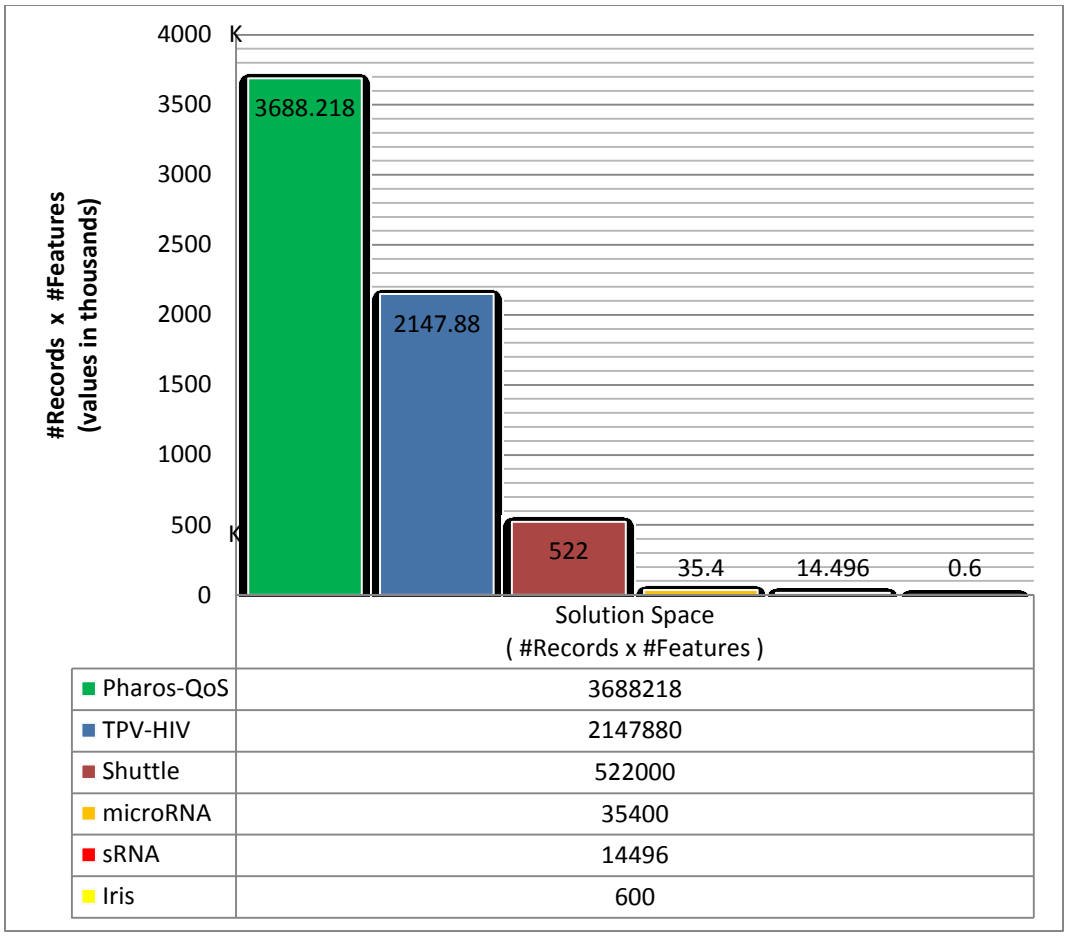


Figure 4 Properties Of Each Dataset Solution Space Used During Experiments.

The datasets contained approximately 10000 to 20000 classified records, including 211 varied (integer and decimal) attributes and 2 possible classifications, each. For the second set of even larger datasets, we are indebted to Momentum Telecom for supplying sample datasets of modem quality of service (QoS) DOCSIS[6] classification data. For the modem QoS datasets, the resultant measurements and evaluation data with each modem classified into a Red/Bad, Yellow/Warning, and Green/Good state was provided. The datasets included up to roughly 500000 multivariate items, each with 9 decimal attributes and 4 possible classifications.

3.4 RESULTS

The initial iteration of the FDT classification tool had a very difficult time dealing with larger datasets. Run times were extremely long and memory errors abounded. For the same configuration settings and datasets, the original FDT 1.0 and the new FDT 2.0 classification tools work comparably (very well) on smaller datasets.

Table 3 FDT 1.0 Versus FDT 2.0 Accuracy Measures.

				FDT 1.0	FDT 2.0
Name	Features	Categories	Size	Accuracy (%)	Accuracy (%)
Pharos-QoS	9	4	XL	(could not run/too large)	96.5
TPV-HIV	210	2	L	(could not run/too large)	99.95
Shuttle	9	7	L	82.44	83.66
microRNA	8	2	M	83.09	82.46
sRNA	8	2	M	50	71.68
Iris	4	3	S	97.22	91.67

The FDT 1.0 classification tool has a difficult time dealing with larger datasets in a timely fashion, and has no way of storing decisions or multiple dataset decisioning models. For larger datasets, FDT 2.0 brings a comparable accuracy level, with the added benefits of having

all of the datasets maintained in a stand-alone database, each dataset now with its own identifiable set of database objects that can be dropped and reused (or maintained, as desired by the tool audience) independently, allowing multiple training/test sets to coexist in the tool. On small datasets, timing is dominated by system overhead (creating database objects, allocating memory, etc) and therefore the algorithmic speedup of FDT 2.0 versus FDT 1.0 cannot be accurately assessed. On the larger datasets where FDT 2.0 improvements could be measured, FDT 1.0 could not run (see Table 3), therefore a reliable estimate of the speedup between the two algorithms could not be determined.

3.5 CONCLUSIONS AND FUTURE WORK

We proposed a new implementation of the freeware fuzzy decision tree induction tool (FDT 2.0), taking advantage of the optimized fuzzy ID3 (FID3) algorithm and integrating its merits with relational database storage and processing capabilities. FDT 2.0 implements a relational database backend to store the test base dataset(s), the fuzzification model(s), and the decision information, from which future decisions can be made without having to rerun the decisioning process. FDT 2.0 has a comparable accuracy level to FDT 1.0, with the added benefits of having the datasets maintained in a stand-alone database, each dataset now with its own identifiable set of database objects independently, allowing multiple datasets to coexist in the tool without interfering with each other. Additionally, the FDT 2.0 tool users can now independently explore the data afterwards via adhoc SQL querying in the resultant relational database, drawing further conclusions from the resultant data, and/or manually pruning data as desired from the datasets.

The optimization of fuzzy decisioning algorithms in order to approximate expert human judgment and disambiguate classifications is incredibly important for working towards the

ability to efficiently work with larger datasets. These larger datasets include multivariate classifications from DNA to Climate Analysis. As the training data is held in a database, it is not necessary to rerun the training steps whenever the application is invoked – with larger datasets, this can be a huge time savings. The FDT classification algorithm outperforms other machine learning algorithms including Random Forest, C4.5, SVM and Knn, and is therefore a prime candidate for integration with a database to facilitate large dataset analysis.

The fuzzy decision tree induction tool (FDT) is still under development. Future directions include increasing the accuracy of predictions, decreasing the speed of predictions, incorporating more automated "learning" elements and working towards decisioning models that can comfortably work with terabyte-sized data sets in a reasonable time frame.

3.6 REFERENCES

[1] N. M. Abu-halaweh, R. W. Harrison, "FDT 1.0: An improved fuzzy decision tree induction tool," Fuzzy Information Processing Society (NAFIPS), 2010 Annual Meeting of the North American , vol., no., pp.1-5, July 2010

[2] C. Z. Janikow, "Fuzzy Decision Trees: Issues and Methods," IEEE Trans. on Man, Systems and Cybernetics, Vol. 28, Issue 1, pp. 1-14, 1998.

[3] Y. Yuan, M. J. Shaw, "Induction of Fuzzy Decision Trees," Fuzzy Sets and Systems, Vol. 69, Issue 2, pp. 125-139, 1995.

[4] UCI Machine Learning Repository, http://archive.ics.uci.edu/ml/.

[5] Genotype-Phenotype Datasets. Stanford University HIV Drug Resistance Database. Retrieved March 12, 2014, from http://hivdb.stanford.edu/cgi-bin/GenoPhenoDS.cgi.

[6] Data Over Cable Service Interface Specifications. Cable Labs. Retrieved Jan 02, 2014, from http://www.cablelabs.com/specifications/CM-SP-PHYv3.0-I08-090121.pdf.

4 A NOVEL APPROACH TO DETERMINE DOCKING LOCATIONS USING FUZZY LOGIC AND SHAPE DETERMINATION

The determination of whether a molecule can bind or "dock" a protein to a certain site depends on the orientation of the molecules, the charge of the atoms that comprise the molecules, and the electrical potential of the proposed area. It is said that a fundamental problem with molecular docking is that the orientation space itself is very large and grows in a combinatorial manner with the number of degrees of freedom of the interacting molecules. We tried to cleverly solve this problem by using shape definitions and fuzzy logic to help reduce the search size of possible docking locations and predict which locations and orientations are the most likely docking locations.

4.1 INTRODUCTION

Molecular docking is a computational procedure that attempts to predict noncovalent binding of macromolecules with other ligands; this binding can affect the behavior of the molecules are around the protein [1] as well as, in organisms, lead to various different effects in that organism whether it is the production of inhibitors or the reproduction of virus. For this reason, the determination of the binding of small molecules to proteins has a very practical application in the field of drug and toxin design [2]. There are many methods for molecular docking and the algorithms to predict binding that is being developed are steadily increasing [3]. Many of these algorithms share common methods with some of the most popular being:

- Molecular Dynamics Methods [4] [5]
- Genetic Algorithms and evolutionary programming [6]
- Fragment-based methods [7] [8]

- Point Complementary Methods [9]
- Quantitative Structure-Activity Relationship (QSAR) Models [10]

Our novel method performs the classification of shapes along each protein and molecule to significantly reduce the search space of likely docking locations by eliminating pairings of unlikely shapes. The cornerstone of our algorithm was the integration of a relational database to hold the atom and protein data, and in the case of small molecule files, using the information it contains to calculate charge. We also calculate the gradient of potential for each shape to measure binding affinity. With the shape information and the gradient of potential, a fuzzy determination is created to determine a ranking system of the possible docking locations. Using a set of 13000 atoms, we proved that our method succeeds in greatly speeding up the process of finding potential docking locations.

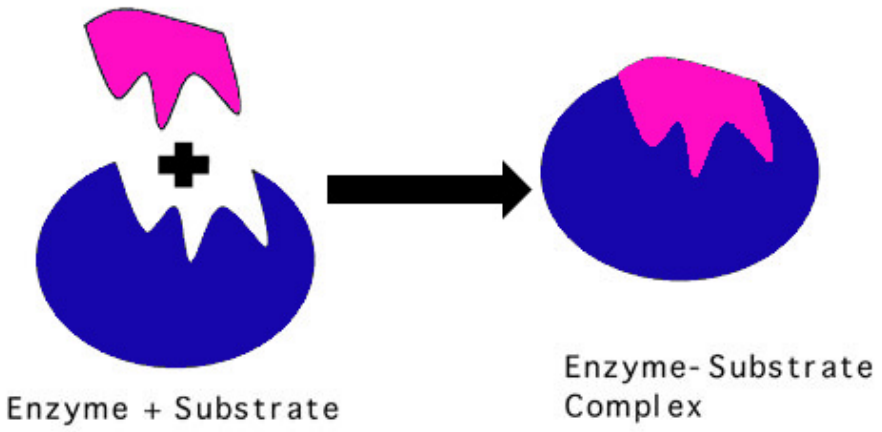


Figure 5 Lock and key protein docking

4.2 ALGORITHM

1) Import the atoms into the database.
2) Add the shape and charge definitions for all proteins.

3) Determine the shape and charge definitions for the small molecule.
4) Cross-reference the two different lists of shapes to generate possible interactions and rank these interactions according to electrostatic favorability.

Table 4 Atoms and Van-Der-Waals radius

Atomic Number	Element Symbol	Van Der Waals Radius
1	H	1.2
6	C	1.7
7	N	1.55
8	O	1.52
9	F	1.47
15	P	1.8
16	S	1.8
–	–	2

Equation 1

$$\theta = \frac{\arccos\left(-s^2 - 2r^2\right)}{2r^2}$$

$$A = \tfrac{2\pi}{\theta}, \qquad B = \tfrac{\pi}{\theta} + 1$$

$$\begin{aligned} atom_{surface} = (&x + r\cos(\theta * a)\sin(\theta * b), \\ &y + r\sin(\theta * a)\sin(\theta * b), \\ &z + r\cos(\theta * b)) \\ \exists a \in [0, A], &\exists b \in [0, B] \end{aligned}$$

Equation 2

$$V_E = \frac{1}{4\pi\epsilon_0}\frac{Q}{r}$$

4.3 SHAPE DETERMINATION

To begin, we extracted key information from standard protein database files (PDB files) [11] into a relational database that we created to hold atom and protein data. We use the energies of the atoms from the protein and molecule entries to create “surfaces" around each object. Geometric shapes are approximated from the imposed surface structure and the shape of the

space around each object. The coordinates of the discovered shapes are saved into the database, then compared to see possible docking locations based on the geometry that is created. This reduces the actual problem space required to find potential docking locations of molecules to

Equation 3

$$\nabla V_E = -\frac{Q}{4\pi\epsilon_0 r^3}\left(x\hat{i} + y\hat{j} + z\hat{k}\right)$$

Equation 4

$$D_E = Qx\hat{i} + Qy\hat{j} + Qz\hat{k}$$

proteins. As surfaces are created on both objects, structures and pockets arise much like in a puzzle: a surface of one object may be able to fit into a pocket of the other. During the initial Discovery step, molecule border definition takes place. We take each atom that comprises the ligand that we are working with and create a surface for each using its Van-Der-Waals radius, as referenced in Table I. Using the radii of the respective atom and a shell spacing constant s we represent the surface of each atom by using a system of x, y, z coordinates, as show in Equation (1). To model the surface, a shell is built around each atom; the radius of the shell is determined by the radius of the element. Next, all of the points on this sphere that overlap a neighboring atom's sphere are absorbed and removed, clumping the atoms together, and the protein is defined as a molecule. The union of all of the atom surfaces represents the complete surface of the ligand as a whole. A sphere is now constructed around this molecule, and probes are extended inwards towards until they touch the ligand surface (Figure 10). These probes are stored in the database as the definition of the shape of the molecule. Individual surface Shape determination is generated by taking collections of adjacent probes. Each collection has its own individual shape, which is determined by the total number of probes in each shape, as well as the maximum

difference along each axis. In order to divide the ligand surface into individual shapes suitable for matching as potential docking sites, each probe is examined and determined whether it is a local minimum (for a Valley) or maximum (for a Peak). If it is not a Peak or a Valley, then the probe is added to a queue, and the next point down is examined in sequence until a known Shape is encountered or a minimum or maximum is reached, defining a new Shape. This process is repeated for all probes until every part of the surface has been explored and assigned to a Shape.

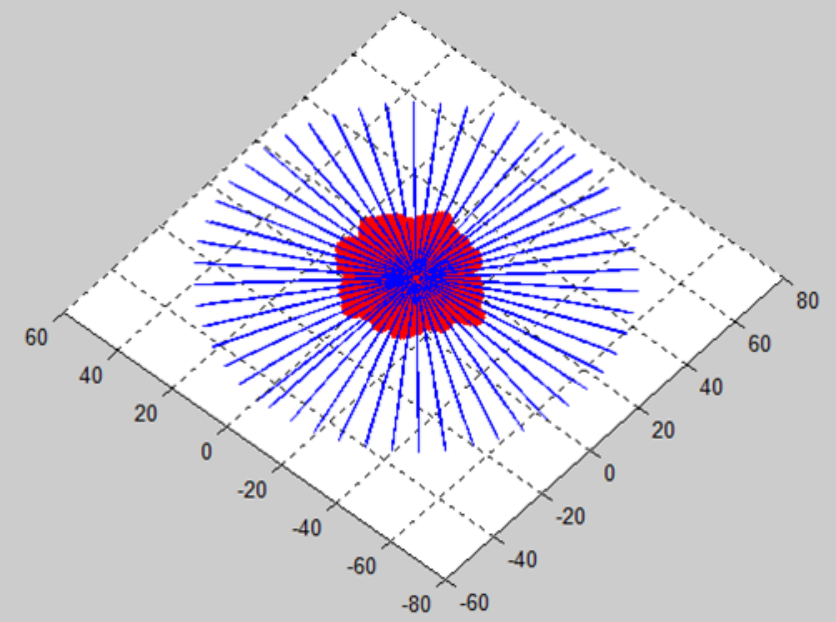


Figure 6 Visualization of the Probe Extension

4.4 MEASURE OF ELECTRICAL POTENTIAL

To calculate if a binding between the two shapes is electrochemically favorable, we measure the electrical potential and the electric dipole potential. Both are used to measure the possible interaction between the respective shape and the others stored in the database. Electric potential is applied to each molecule as a whole as well as the shapes that are created by them. The calculation is done by taking a ration of a point charge Q and a distance from the atoms, r,

ϵ_0

with representing the Coulomb potential, which is an electrical constant measuring the electric potential of free space. The calculation of the electric dipole moment in Equation (4) is taken to determine the strength of the charge as well as overall polarity of the shape. The potential of an electrical dipole is found by superposing the point charge potentials of the charges. The calculation we used assumes that the system has an overall neutral charge.

Equation 5

$$SS\left(p,m\right)=\left(1-\frac{w_p-w_m}{w_p}\right)*\left(1-\frac{d_p-d_m}{d_p}\right)$$

Equation 6

$$BA\left(p,m\right)=\frac{C_p\cdot C_m}{max\left(\left\|C_m,C_p\right\|\right)^2}$$

Equation 7

$$f_d\left(p,m\right)=\frac{SS_{p,m}\left(BA\left(p,m\right)+1\right)}{2}$$

4.5 FUZZY LOGIC

The fuzzy logic is applied to both the shape and the electrical potential separately; then, using standard fuzzy logic, the membership values are applied to a logical AND operation by multiplying the two values together. For simplicity, we name the membership values for shape fitting and electrical potential the fit value and the affinity value respectively. The fit value is calculated by comparing the overall volume, size, depth, and width w of the shapes. From this we calculate a shape similarity by Equation (5). If the size of the docking molecule is ever overly

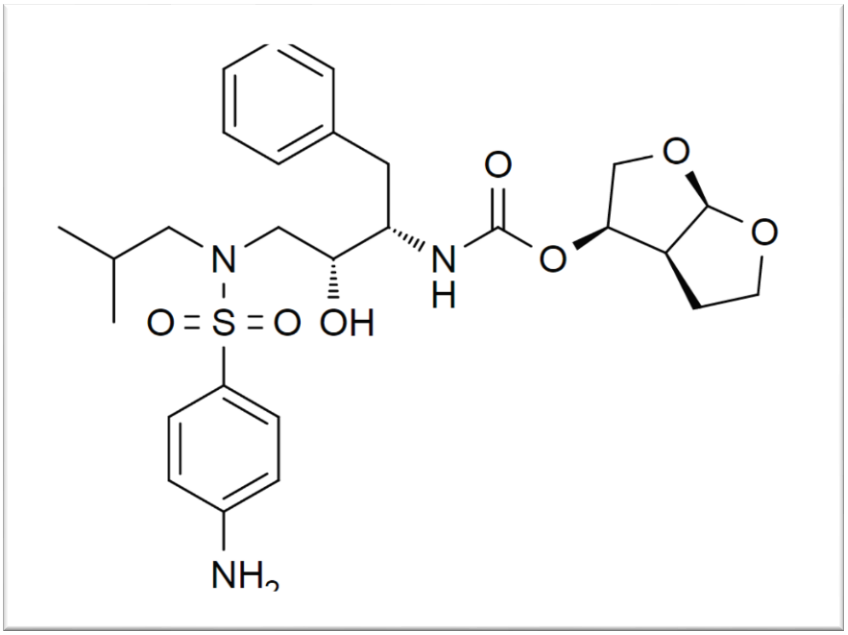


Figure 7 Structure of Darunavir [12]

large for a prospective pocket shape, it is automatically given a membership value of zero which effectively cancels the likelihood of membership. A comparison of similarity is then taken and multiplied together to get total fuzzy membership value. The dot product of the vector of the charges of the molecule m and protein p are calculated to determine the fuzzy membership value for binding affinity (Equation (6)). Ideally, the two charges will be equal and opposite, meaning that perpendicular charges would be unfavorable. From this logic we can then calculate the binding affinity membership. When we combine shape similarity and binding affinity calculations we will calculate our overall membership value for binding. The Fuzzy Determination calculation in Equation (7) weights the shape more heavily in the overall membership function.

4.6 CONCLUSIONS AND FUTURE WORK

The current version of the algorithm uses a single probe per shape, comparison measures for shape similarity to consider the width, height and depth of a potential docking location, and produces good results in terms of speed of return. Running the Darunavir molecule against the shapes that correlated to the each of the 7 proteins in the database completed in 27.644 seconds, an average of roughly 4 seconds per protein when searching through all feasible docking locations on each protein. Natural usage and growth in our database will cause it to become a repository of similar data: in the future it would also be desirable to use machine learning techniques to find functional groups of proteins using shared shape information, as geometry plays a large role in protein function and the determination and classification of functional groups of proteins is a large open area in the bioinformatics community.

4.7 REFERENCES

[1] M. O. Taha, Y. Bustanji, A. G. Al-Bakri, A.-M. Yousef, W. A. Zalloum, I. M. Al-Masri, and N. Atallah, “Discovery of new potent human protein tyrosine phosphatase inhibitors via pharmacophore and qsar analysis followed by in silico screening," Journal of Molecular Graphics and Modelling, vol. 25, no. 6, pp. 870–884, 2007.

[2] O. Trott and A. J. Olson, “Autodock vina: improving the speed and accuracy of docking with a new scoring function, efficient optimization, and multithreading," Journal of computational chemistry, vol. 31, no. 2, pp. 455–461, 2010.

[3] R. D. Taylor, P. J. Jewsbury, and J. W. Essex, “A review of protein small molecule docking methods," Journal of computer-aided molecular design, vol. 16, no. 3, pp. 151–166, 2002.

[4] D. A. Pearlman, D. A. Case, J. W. Caldwell, W. S. Ross, T. E. Cheatham III, S. DeBolt, D. Ferguson, G. Seibel, and P. Kollman, “Amber, a package of computer programs for applying molecular mechanics, normal mode analysis, molecular dynamics and free energy calculations to simulate the structural and energetic properties of molecules," Computer Physics Communications, vol. 91, no. 1, pp. 1–41, 1995.

[5] B. R. Brooks, R. E. Bruccoleri, B. D. Olafson, D. J. States, S. Swaminathan, and M. Karplus, “Charmm: A program for macromolecular energy, minimization, and dynamics calculations," Journal of computational chemistry, vol. 4, no. 2, pp. 187–217, 1983.

[6] G. Jones, P. Willett, R. C. Glen, A. R. Leach, and R. Taylor, “Development and validation of a genetic algorithm for flexible docking," Journal of molecular biology, vol. 267, no. 3, pp. 727–748, 1997.

[7] T. J. Ewing and I. D. Kuntz, “Critical evaluation of search algorithms for automated molecular docking and database screening," Journal of Computational Chemistry, vol. 18, no. 9, pp. 1175–1189, 1997.

[8] B. Kramer, G. Metz, M. Rarey, and T. Lengauer, “Part 1–docking and scoring: Methods development-ligand docking and screening with flexx,"Medicinal Chemistry Research, vol. 9, no. 7-8, pp. 463–478, 1999.

[9] H. A. Gabb, “Ftdock (v1. 0)," 1997.

[10] A. Z. Dudek, T. Arodz, and J. Galvez, “Computational methods in developing quantitative structure-activity relationships (qsar): a review,"Combinatorial chemistry & high throughput screening, vol. 9, no. 3, pp.213–228, 2006.

[11] “PDB File Format - Contents Guide (3.30)," World Wide Web, http://www.wwpdb.org/documentation/format33/v3.3.html.

[12] B. Horne, “Structure of darunavir,"

https://upload.wikimedia.org/wikipedia/commons/0/07/Darunavir2DCSD.svg.

A Novel Approach to Determine Docking Locations Using Fuzzy Logic and Shape Determination

Chinua Umoja, J.T. Torrance, Erin-Elizabeth A. Durham, Andrew Rosen, Dr. Robert Harrison
Computer Science Department, Georgia State University, Atlanta, GA 30303

ABSTRACT

The determination of whether a molecule can bind or "dock" a protein to a certain site depends on the orientation of the molecules, the charge of the atoms that comprise the molecules, and the electrical potential of the proposed area. It is said that a fundamental problem with molecular docking is that the orientation space itself is very large and grows in a combinatorial manner with the number of degrees of freedom of the interacting molecules. We tried to cleverly solve this problem by using shape definitions and fuzzy logic to help reduce the search size of possible docking locations and predict which locations and orientations are the most likely docking locations.

INTRODUCTION

The prediction of where a molecule can bind to a protein is a very important area of study in biology, specifically regarding determining the function of proteins, as well as in drug and toxin design. Computational prediction of the binding of small molecules to proteins is a critical first step in the development of novel drugs and inhibitors. While many approaches for predicting binding have been developed and are widely used in computational chemistry, there is room for significant improvement both in computational performance and prediction accuracy.

Molecular Docking is a computational procedure that attempts to predict the non-covalent binding of macromolecules with other ligands. This binding can affect the behavior of the molecules around the protein as well as, in organisms, lead to various different effects in that organism - whether it is the production of inhibitors or the reproduction of a virus. The problem statement addresses the issues of search space, as well as the technical issues of determining the orientation of the two molecules in three dimensions, and the chemical composition and their impact on the electrical charges of the atoms that comprise each molecule.

Our aim is to develop a very efficient algorithm for rapid screening of many small molecules, that is still reasonably accurate and suitable for data mining of large chemical databases. We propose a method of searching for possible docking locations between a small molecule and a protein by determining and storing the unique shape conformations of a molecule into a relational database, and then determining and storing into the database the charge information for each shape. Our method runs a comparison on the shape and charge information, which is used to create a fuzzy measure that ranks the likelihood of a successful docking attempt.

ALGORITHM

1. Import the atoms into the database.
2. Add the shape and charge definitions for all proteins.
3. Determine the shape and charge definitions for the small molecule.
4. Cross-reference the two different lists of shapes to generate possible interactions, and rank these interactions according to electrostatic favorability.

Figure 1: Visual representation of 3TTP

SHAPE DETERMINATION

To begin, we extracted key information from standard protein database files (PDB files) into a relational database that we created to hold atom and protein data. We use the energies of the atoms from the protein and molecule entries to create "surfaces" around each object. Geometric shapes are approximated from the imposed surface structure and the shape of the space around each object. The coordinates of the discovered shapes are saved into the database, then compared to see possible docking locations based on the geometry that is created. This reduces the actual problem space required to find potential docking locations of molecules to proteins. As surfaces are created on both objects, structures and pockets arise much like in a puzzle: a surface of one object may be able to fit into a pocket of the other.

Atomic Number	Element Symbol	Van Der Waals Radius
1	H	1.2
6	C	1.7
7	N	1.55
8	O	1.52
9	F	1.47
15	P	1.8
16	S	1.8
...	...	2

Table 1: Atoms and Van-Der-Waals radius

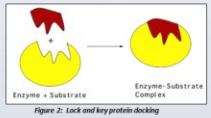


Figure 2: Lock and key protein docking

During the initial Discovery step, molecule border definition takes place. We take each atom that comprises the ligand and create a surface using its Van-Der-Waals radius, as referenced in *Table 1*. Using the radii of the respective atom and a shell spacing constant s, we represent the surface of each atom by using a system of x, y, and z coordinates.

To model the surface, a shell is built around each atom; the radius of the shell is determined by the radius of the element. Next, all of the points on this sphere that overlap a neighboring atoms sphere are absorbed and removed, clumping the atoms together, and the protein is defined as a molecule. The union of all of the atom surfaces represents the complete surface of the ligand as a whole.

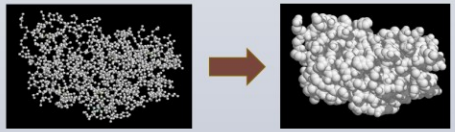


Figure 3: Ball and stick representation of 3TTP (left). Space fill representation of 3TTP (right).

A sphere is now constructed around this molecule, and probes are extended inwards towards until they touch the ligand surface. These probes are stored in the database as the definition of the shape of the molecule.

Individual surface Shape determination is generated by taking collections of adjacent probes. Each collection has its own individual shape, which is determined by the total number of probes in each shape, as well as the maximum difference along each axis. In order to divide the ligand surface into individual shapes suitable for matching as potential docking sites, each probe is examined and determined whether it is a local minimum (for a Valley) or maximum (for a Peak).

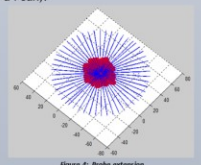


Figure 4: Probe extension

If it is not a Peak or a Valley, then the probe is added to a queue, and the next point down is examined in sequence until a known Shape is encountered or a minimum or maximum is reached, defining a new Shape. This process is repeated for all probes until every part of the surface has been explored and assigned to a Shape.

MEASURE OF ELECTRICAL POTENTIAL

To calculate if a binding between the two shapes is electrochemically favorable, we measure the electrical potential and the electric dipole potential. Both are used to measure the possible interaction between the respective shape and the others stored in the database.

Electric potential is applied to each molecule as a whole, as well as the shapes that are created by them. The calculation is done by taking a ratio of a point charge Q, and a distance from the atoms, r, with ε_0 representing the Coulomb potential, which is an electrical constant measuring the electric potential of free space.

$$V = \sum_{i=1}^{N} \frac{1}{4\pi\varepsilon_0}\left(\frac{q_i}{r_i}\right)$$

Eq. 1: Electrostatic potential

$$\begin{aligned}\mathbf{p}(\mathbf{r}) &= \sum_{i=1}^{N} \int_V q_i[\delta(\mathbf{r}_0 - (\mathbf{r}_i + \mathbf{d}_i)) - \delta(\mathbf{r}_0 - \mathbf{r}_i)](\mathbf{r}_0 - \mathbf{r})\, d^3\mathbf{r}_0 \\ &= \sum_{i=1}^{N} q_i\,[\mathbf{r}_i + \mathbf{d}_i - \mathbf{r} - (\mathbf{r}_i - \mathbf{r})] \\ &= \sum_{i=1}^{N} q_i\mathbf{d}_i = \sum_{i=1}^{N} \mathbf{p}_i\,.\end{aligned}$$

Eq. 2: Electric dipole moment

The calculation of the electric dipole moment is taken to determine the strength of the charge as well as overall polarity of the shape. The potential of an electrical dipole is found by superposing the point charge potentials of the charges. The calculation we used assumes that the system has an overall neutral charge.

FUZZY LOGIC

The fuzzy logic is applied to both the shape and the electrical potential separately; then, using standard fuzzy logic, the membership values are applied to a logical "and" operation by multiplying the two values together. For simplicity, we name the membership values for shape fitting and electrical potential the "fit" value and the "affinity" value respectively.

The "fit" value is calculated by comparing the overall volume, size, depth, and width of the shapes. If the size of the docking molecule is ever overly large for a prospective pocket shape, it is automatically given a membership value of zero which effectively cancels the likelihood of membership. A comparison of similarity is then taken and multiplied together to get total fuzzy membership value.

$$SS(p,m) = \left(1 - \frac{w_p - w_m}{w_p}\right) * \left(1 - \frac{d_p - d_m}{d_p}\right)$$

Eq. 3: Shape Similarity: where p is the protein, m is the molecule, w represent width and d represents depth, respectively, and SS(p,m) denotes the shape similarity of the protein and the molecule.

$$BA(p,m) = \frac{C_p \cdot C_m}{max\left(\|C_m, C_p\|\right)^2}$$

Eq. 4: Binding Affinity: where p is the protein, m is the molecule, C denotes the electrostatic potential and BA(p,m) denotes the binding affinity of the protein and the molecule.

$$f_d(p,m) = \frac{SS_{p,m}\,(BA(p,m)+1)}{2}$$

Eq. 5: Overall "fit": fuzzy membership value for binding

CONCLUSIONS

The current version of the algorithm uses a single probe per shape, comparison measures for shape similarity to consider the width, height and depth of a potential docking location, and produces good results in terms of speed of return. Running the Darunavir molecule against the shapes that correlated to the each of the 7 proteins in the database completed in 27.644 seconds, an average of roughly 4 seconds per protein when searching through all feasible docking locations on each protein.

Natural usage and growth in our database will cause it to become a repository of similar data: in the future it would also be desirable to use machine learning techniques to find functional groups of proteins using shared shape information, as geometry plays a large role in protein function and the determination and classification of functional groups of proteins is a large open area in the bioinformatics community.

Figure 8 **A Novel Approach to Determining Docking Locations Using Fuzzy Logic and Shape Determination (Poster)**

5 A NOVEL APPROACH TO DETERMINE DOCKING LOCATIONS USING FUZZY SHAPE RECOGNITION

Computational prediction of the binding of small molecules to proteins is a critical first step in the development of novel drugs and inhibitors. While many approaches for predicting binding have been developed and are widely used in computational chemistry, there is room for significant improvement both in computational performance and prediction accuracy. The aim of this work is to develop a very efficient algorithm for rapid screening of many small molecules, that is still reasonably accurate and suitable for data mining of chemical databases. In this paper we propose a method of searching for possible docking locations between a small molecule and a protein by determining and storing the unique shape conformations of a molecule and determining the charge information for each shape. Our method runs a comparison on the shapes and charge information which is used to create a fuzzy measure then ranks the likelihood of docking.

5.1 INTRODUCTION

Molecular docking is a computational procedure that attempts to predict noncovalent binding of macromolecules with other ligands; this binding can affect the behavior of the molecules are around the protein [9] as well as, in organisms, lead to various different effects in that organism whether it is the production of inhibitors or the reproduction of virus. For this reason, the determination of the binding of small molecules to proteins has a very practical application in the field of drug and toxin design [11]. There are many methods for molecular docking and the algorithms to predict binding that is being developed are steadily increasing [10]. Many of these algorithms share common methods with some of the most popular being:

- Genetic Algorithms and evolutionary programming [5]
- Fragment-based methods [3] [6]
- Point Complementary Methods [4]
- Quantitative Structure-Activity Relationship (QSAR) Models [2]

Our novel method performs the classification of shapes along each protein and molecule to significantly reduce the search space of likely docking locations by eliminating pairings of unlikely shapes. The cornerstone of our algorithm was the integration of a relational database to hold the atom and protein data, and in the case of small molecule files, using the information it contains to calculate charge. We also calculate the gradient of potential for each shape to measure binding affinity. With the shape information and the gradient of potential, a fuzzy determination is created to determine a ranking system of the possible docking locations. Using a set of 13000 atoms, we proved that our method succeeds in greatly speeding up the process of finding potential docking locations.

5.2 METHODOLOGY

Extracting the key information from protein database (PDB [12]) files into a relational database of atom and protein data, and in the case of small molecule files, using the source file information provided in the PDB files to pre-calculate and storing charge information, allowed us to build a reusable datastore of atomic level information about the proteins, shape information about the proteins (including physical and chemical information), and small molecule shape information. Each atom is given a radius directly related to its Van-Der Waals measure, its

respective x, y, and z coordinates, and positive or negative charge information. Each calculation was only performed once and then stored into the database to be recalled as needed for comparison.

```
:: FORMAT ::
COLUMNS        DATA  TYPE    FIELD        DEFINITION
---------------------------------------------------------------------------------
 1 -  6        Record name   "ATOM  "
 7 - 11        Integer       serial       Atom  serial number.
13 - 16        Atom          name         Atom name.
17             Character     altLoc       Alternate location indicator.
18 - 20        Residue name  resName      Residue name.
22             Character     chainID      Chain identifier.
23 - 26        Integer       resSeq       Residue sequence number.
27             AChar         iCode        Code for insertion of residues.
31 - 38        Real(8.3)     x            Orthogonal coordinates for X in Angstroms.
39 - 46        Real(8.3)     y            Orthogonal coordinates for Y in Angstroms.
47 - 54        Real(8.3)     z            Orthogonal coordinates for Z in Angstroms.
55 - 60        Real(6.2)     occupancy    Occupancy.
61 - 66        Real(6.2)     tempFactor   Temperature  factor.
77 - 78        LString(2)    element      Element symbol, right-justified.
79 - 80        LString(2)    charge       Charge  on the atom (ignored).

:: Sample Records ::
0         1         2         3         4         5         6         7         8
01234567890123456789012345678901234567890123456789012345678901234567890123456789
ATOM   5002  CG  TRP B 416      23.292 -16.836 -25.778  1.00 17.45           C
HETATM11295  O   HOH A2059      37.126 -15.822 -97.534  1.00  8.80           O
```

Figure 9 PDB Source File Format

5.3 ALGORITHM

1) Import the atoms into the database.
2) Add the shape and charge definitions for all proteins.
3) Determine the shape and charge definitions for the small molecule.
4) Cross-reference the two different lists of shapes to generate possible interactions and rank these interactions according to electrostatic favorability.

5.4 IDENTIFICATION

During the initial Discovery step, molecule border definition takes place. We take each atom that comprises the ligand that we are working with and create a surface for each using its Van-Der-Waals radius. Using the radii of the respective atom and a shell spacing constant s we represent the surface of each atom by using a system of x, y, z coordinates, as referenced in Equation 8. To model the surface, a shell is built around each atom; the radius of the shell is determined by the radius of the element. Next, all of the points on this sphere that overlap a neighboring atom's sphere are absorbed and removed, clumping the atoms together, and the protein is defined as a molecule. The union of all of the atom surfaces represents the complete surface of the ligand as a whole.

A sphere is now constructed around this molecule, and probes are extended inwards towards until they touch the ligand surface. These probes are stored in the database as the definition of the shape of the molecule.

Equation 8

$$\theta = \frac{\arccos\left(-s^2 - 2r^2\right)}{2r^2}$$

$$A = \tfrac{2\pi}{\theta}, \quad B = \tfrac{\pi}{\theta} + 1$$

$$atom_{surface} = \left(x + r\cos\left(\theta * a\right)\sin\left(\theta * b\right), y + r\sin\left(\theta * a\right)\sin\left(\theta * b\right), z + r\cos\left(\theta * b\right)\right)$$

$$\exists a \in [0, A], \exists b \in [0, B]$$

In order to divide the ligand surface into individual shapes suitable for matching as potential docking sites, each probe is examined and determined whether it is a local minimum (for a Valley) or maximum (for a Peak). If it is not one of the above it is added to a queue, and the next point down is examined in sequence until a known Shape is encountered or a minimum or maximum is reached, defining a new Shape. This process is repeated for all probes until every parts of the surface has been explored and assigned to a Shape. It should be noted that this

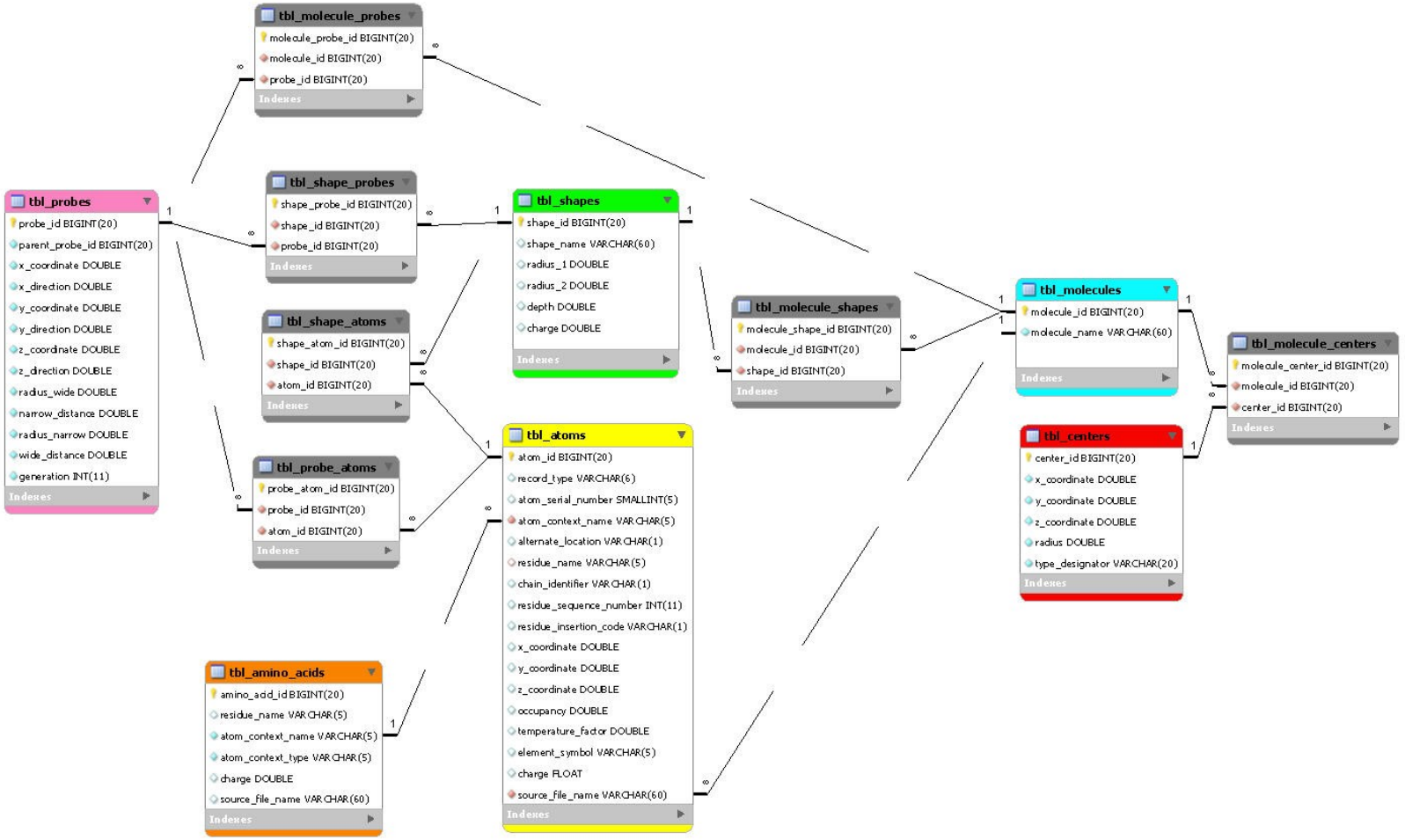


Figure 10 Relational database to hold the atom and protein data

is a one-to-one mapping, and in this iteration of the software, that each probe can owned by a single shape. The atoms that compose this shape are then determined by proximity to the probes it is comprised of. The shape is defined by four physical parameters: height, number of member atoms, radius-1, and radius-2. The shape's height is determined by the difference between the height of the longest probe and that of the shortest probe in the shape. To determine the two radii of the shape, two transformations are done so that the center point of the shape rests on the z-axis. The length of the shape in the x and y directions are then found and the smaller size is assigned to radius-1 and the larger to radius-2. These parameters are what are used to physically compare shapes for fit.

Table 5 Atoms and Van-Der-Waals radius

Atomic Number	Element Symbol	Van Der Waals Radius
1	H	1.2
6	C	1.7
7	N	1.55
8	O	1.52
9	F	1.47
15	P	1.8
16	S	1.8
–	–	2

During shape creation we take each atom that comprises the ligand that we are working with and create a surface for each using its Van-Der-Waals radius (Table 5). Using the radii of the respective atom and a shell spacing constant s we represent the surface of each atom by using a system of x, y, z coordinates, as referenced in Equation 8. The union of all of the atom surfaces is what we use to represent the surface of the ligand as a whole.

Equation 9

$$protein_{surface} = \bigcup_{atom \in protein} atom_{surface}$$

Equation 10

$$shell_{center} = \left(\frac{x_{min} + x_{max}}{2}, \frac{y_{min} + y_{max}}{2}, \frac{z_{min} + z_{max}}{2} \right)$$

$$shellradius = max\sqrt{(x - x_c)^2 + (y - y_c)^2 + (z - z_c)^2} + C$$

$$shell_{center} = (x_c, y_c, z_c)$$

Equation 11

$$probe_{direction} = \left(\frac{x + x_c}{shell_{radius}}, \frac{y + y_c}{shell_{radius}}, \frac{z + z_c}{shell_{radius}} \right)$$

After the ligands surface is ascertained, a search for significant shapes on its surface is performed. We do this search by creating a spherical perimeter around the ligand based on the three dimensional center of ligand as well as a radius from the center that is formed by finding the furthest distance between all points in the ligands surface plus a constant for a buffer space. We use each point that comprises the perimeter as a starting location for a collection of probes that move toward the center of the molecule. This is done by projecting the direction vector to travel between the point on the perimeter and the center. We repeatedly subtract the direction vector from each point on the perimeter until there is a point on the surface that is at most an angstrom away or to the point where the length of this probe would be greater than or equal to the diameter of the perimeter sphere indicating this probe had traveled through an empty hole in the protein and would be removed from the next portion of the algorithm. From these probes we calculate the shape determinations we use to determine docking locations.

The shapes are determined by finding what we define as peaks and valleys: Valleys are collections of probes that surround a probe of maximum length, as well as any probes that are adjacent to a probe already in the valley that are of a smaller length. Peaks are collections of probes that surround a probe of minimum length, as well as any probes that are adjacent to a

probe already in the valley that are of a larger length. Each peak/valley is stored based on its number of probes, depth, width and height; height being defined as the difference between the longest probe and the shortest probe, while depth and width are determined by rotating the shape along the z axis and storing the length of the longest x and y axis.

Equation 12

$$E = \sum_{atoms} V_E$$

$$V_E = \frac{1}{4\pi\epsilon_0}\frac{Q}{r}$$

$$r = \sqrt{(x-x_c)^2 + (y-y_c)^2 + (z-z_c)^2} + C$$

$$\nabla V_E = \frac{\delta V_E}{\delta x}\hat{i} + \frac{\delta V_E}{\delta y}\hat{j} + \frac{\delta V_E}{\delta z}\hat{k} = \frac{\delta V_E}{\delta r}\frac{\delta r}{\delta x}\hat{i} + \frac{\delta V_E}{\delta r}\frac{\delta r}{\delta y}\hat{j} + \frac{\delta V_E}{\delta r}\frac{\delta r}{\delta z}\hat{k}$$

$$\frac{\delta V_E}{\delta r} = \frac{1}{4\pi\epsilon_0}\frac{Q}{r^2}$$

$$\frac{\delta r}{\delta x} = \frac{x-x_c}{r}, \frac{\delta r}{\delta y} = \frac{y-y_c}{r}, \frac{\delta r}{\delta z} = \frac{z-z_c}{r}$$

$$\nabla V_E = -\frac{Q}{4\pi\epsilon_0 r^3}\left(x\hat{i} + y\hat{j} + z\hat{k}\right)$$

$$\nabla E = \sum_{atoms} \nabla V_E$$

5.5 CHARGE DEFINITION

For each shape in the ligand, a charge definition is assigned. The charge definition is dependent on whether the ligand is a protein or a small molecule.: If the shape is on the surface of a protein the charge is defined as the gradient of the electrostatic potential inside the shape [4]. The electrostatic potential and the derivation of its gradient are shown in Equation 13, where Q is

the specific atom charge (sourced from a PDB file) and the electrical constant, 322.17752 kcal/mol per angstrom. The gradient of potential was calculated for each shape based on the atoms that comprised it. This results in the potential in being in the form of Kcal/mol.

Equation 13

$$SS(p,m) = \left(1 - \frac{w_p - w_m}{w_p}\right) * \left(1 - \frac{d_p - d_m}{d_p}\right)$$

Equation 14

$$BA(p,m) = \frac{C_p \cdot C_m}{max\left(\|C_m, C_p\|\right)^2}$$

Equation 15

$$f_d(p,m) = \frac{SS_{p,m}\left(BA(p,m)+1\right)}{2}$$

For the charge definition of a shape on the surface of a smaller molecule, the dipole moment was used. The calculation of the dipole moment uses the entire molecular structure of the small molecule and is based on the assumption of an overall neutral charge. Because there is no standard definition for the charge of an atom it had to be computed based on the known structure, for this SDF files were used. SDF files are used by chemists and define what atoms are in a molecule and what bonds exist between these atoms. This information is fed into an existing program, AMMP, to determine the special position of the atoms with their bonds as well as the charge on each atom due to positioning. The data from this is used to calculate the dipole moment of the small molecules based on Equation 16. This electric potential is stored in the

Equation 16

$$D_E = Qx\hat{i} + Qy\hat{j} + Qz\hat{k}$$

database along with the corresponding shape to be called when the fuzzy logic and comparison rankings are applied.

5.6 RANKING

The first stage of the ranking system contains a pruning step that eliminates all shapes that correlate to peaks form small molecules that are too large for valleys of the protein. We do this by comparing depth, width and height to the two respective shapes. From this we calculate a shape similarity by Equation 13.

The dot product of the vector of the charges of the molecule and protein are calculated to determine the fuzzy membership value for binding affinity. Ideally, the two charges will be equal and opposite, meaning that perpendicular charges would be unfavorable. From this logic we can then calculate the binding affinity membership. When we combine shape similarity and binding affinity calculations we will calculate our overall membership value for binding. The Fuzzy Determination calculation weights the shape more heavily in the overall membership function.

5.7 RESULTS

For our algorithm we were not only interested in the accuracy of the results, but also how swiftly those results could be attained. Our goal was to the overall speed of our algorithm as well as the effectiveness of our shape similarity, binding affinity and fuzzy measures. In testing we used the Darunavir molecule, a 75 atom compound. Darunavir is a protein inhibitor known to bind with HIV-1 protease (PDB code 3TTP): the flexibility and overall speed of the algorithm was demonstrated in by running directly against Darunvir, versus just an individual protein as

most docking algorithms do. In addition, we decided to run it against all eight possible proteins in our database at the same time (3TTP, 3OXC,5CHA, 3TKW, 4DQC, 1CHG, and 3UFN).

Table 6 Top Results

MDB ID	AA	Residue #	Ss(p,m)	Ba(p,m)	f_d (p,m)
3OXC	ASP	160	0.872433214	0.222370164	0.533218166
3TTP	MET	46	0.959990745	0.102672443	0.52927767
3TTP	GLN	92	0.864394472	0.216091439	0.525591359
3UFN	ARG	57	0.87642945	0.185890614	0.519674729
3OXC	ILE	166	0.863601159	0.202379828	0.519188307
3TKW	PRO	44	0.901837759	0.136108968	0.512292983
3TKW	ASP	60	0.793681923	0.285039546	0.509956329
2PKA	LEU	235	0.998250039	0	0.499125019
2PKA	LEU	108	0.994662513	0	0.497331256
3OXC	LEU	123	0.954290608	0.030470563	0.49168419

The algorithm took 112.388 seconds to find the shapes that correlated to Darunavir and to store each shape, along with its charge information, into the database, with an average time of 4.5 seconds per shape. After all shapes were found, the tool took 27.644 seconds to compare the shape of the Darunavir against the shapes that correlated to the each of the 8 proteins in the database, from which 553 potential matches (hits) were returned.

Table 7 Top ten results based on fuzzy membership values

MDB ID	AA	Residue #	Ss(p,m)	Ba(p,m)	f_d (p,m)
3TKW	GLN	61	0.06214061	0.961263387	0.060937051
3TKW	LYS	45	0.073144085	0.922902992	0.07032449
3OXC	THR	196	0.107108736	0.808699771	0.096863773
4DQC	TRP	6	0.127841634	0.795284156	0.11475603
3OXC	ALA	128	0.04927735	0.769947463	0.04360916
3OXC	PRO	9	0.072587562	0.759986237	0.063876555
3TTP	ASN	98	0.218196564	0.759423495	0.191950081
4DQC	ARG	87	0.05298619	0.758817716	0.046596525
4DQC	ILE	62	0.19828912	0.75768409	0.174264816
3OXC	ASP	130	0.087277074	0.750739213	0.076399698

Table 6 records the top 10 results from the testing against the Darunavir molecule, a 75 atom compound. In Tables 7 and 8, the results are sorted and ranked by the top ten results for

binding affinity and shape similarity (best fuzzy membership value) respectively. When ranked by best fuzzy membership value, of the top ten results, eight are proteins that Darunavir is known to interact with, while the other two results are the same protein 2PKA. Interestingly enough, the binding affinity for the 2PKA result had a membership value of zero - which implies that while the shape of the pocket is very favorable, the binding affinity would make it too difficult to actually dock at that location. Between the results sorted by binding affinity and shape similarity, the same proteins are highly ranked... but because of the low shape similarity scores the fuzzy membership values are still low.

Table 8 Top ten results based on binding

Table 4: Top 10 Shape Binding Affinities

MDB ID	AA	Residue #	Ss(p,m)	Ba(p,m)	f_d (p,m)
2PKA	LEU	235	0.998250039	0	0.499125019
2PKA	LEU	108	0.994662513	0	0.497331256
2PKA	GLY	188	0.980740874	0	0.490370437
3TTP	MET	46	0.959990745	0.102672443	0.52927767
3OXC	LEU	123	0.954290608	0.030470563	0.49168419
2PKA	ILE	200	0.929318268	0	0.464659134
5CHA	GLY	187	0.926654171	0	0.463327086
3TKW	ARG	87	0.913662621	0.05274973	0.480929038
5CHA	PRO	4	0.906068794	0	0.453034397
3TKW	PRO	44	0.901837759	0.136108968	0.512292983

5.8 CONCLUSIONS AND FUTURE WORK

The current version of the algorithm uses a single probe per shape, comparison measures for shape similarity consider the width, height and depth of a potential docking location, and produces good results in terms of speed of return. Running the Darunavir molecule against the shapes that correlated to the each of the 7 proteins in the database completed in 27.644 seconds, an average of roughly 4 seconds per protein when searching through all feasible docking locations on each protein. But in future iterations a measure of suitability could be also created

by determining the alignment of the probes to see if the differences in probe-length for a shape lines-up well with each other. In terms of the binding affinity, future work will include factoring in hydrophobicity, which is said to be highly important in terms of docking [8]

Improvements in shape determination can be made to refine more actual definition inside of the pocket (the potential docking location). In the future, we would also like to adjust the program to account for protein to protein interactions which currently is not a larger step to take since we are already storing the pockets of all proteins we are using; we would only need to also store "peak" information or possible binding regions that could bind to other proteins pockets. Natural usage and growth in our database will cause it to become a repository of similar data: in the future it would also be desirable to use machine learning techniques to find functional groups of proteins using shared shape information, seeing that geometry plays a large role in protein function and that the determination and classification of functional groups of proteins is a large open area in the bioinformatics community.

Future directions will include refining the algorithm to include protein:protein shape definitions: Amino acids are the building blocks of proteins and they are also comprised of atoms. The center of the amino acid is the alpha carbon, which would be the new reference point. Because each amino acid has an electric potential the next version of our algorithm will be tasked with focusing around the coordinate values of the alpha carbon of each amino acid instead of each atom.

5.9 REFERENCES

[1] Bernard R Brooks, Robert E Bruccoleri, Barry D Olafson, David J States, S Swaminathan, and Martin Karplus. Charmm: A program for macromolecular energy,

minimization, and dynamics calculations. Journal of computational chemistry, 4(2):187{217, 1983.

[2] Arkadiusz Z Dudek, Tomasz Arodz, and Jorge Galvez. Computational methods in developing quantitative structure-activity relationships (qsar): a review. Combinatorial chemistry & high throughput screening, 9(3):213{228, 2006.

[3] Todd JA Ewing and Irwin D Kuntz. Critical evaluation of search algorithms for automated molecular docking and database screening. Journal of Computational Chemistry, 18(9):1175{1189, 1997.

[4] Henry A Gabb. Ftdock (v1. 0), 1997.

[5] Gareth Jones, Peter Willett, Robert C Glen, Andrew R Leach, and Robin Taylor. Development and validation of a genetic algorithm for flexible docking. Journal of molecular biology,267(3):727{748, 1997.

[6] B Kramer, G Metz, M Rarey, and T Lengauer. Part 1{docking and scoring: Methods development-ligand docking and screening with flexx. Medicinal Chemistry Research, 9(7-8):463{478, 1999.

[7] David A Pearlman, David A Case, JamesWCaldwell, Wilson S Ross, Thomas E Cheatham III,Steve DeBolt, David Ferguson, George Seibel, and Peter Kollman. Amber, a package of computer programs for applying molecular mechanics, normal mode analysis, molecular dynamics and free energy calculations to simulate the structural and energetic properties of molecules. Computer Physics Communications, 91(1):1{41, 1995.

[8] Alberte Pullman and Bernard Pullman. Molecular electrostatic potential of the nucleic acids. Quarterly reviews of biophysics, 14(03):289{380, 1981.

[9] Mutasem O Taha, Yasser Bustanji, Amal G Al-Bakri, Al-Motassem Yousef, Waleed A Zalloum, Ihab M Al-Masri, and Naji Atallah. Discovery of new potent human protein tyrosine phosphatase inhibitors via pharmacophore and qsar analysis followed by in silico screening. Journal of Molecular Graphics and Modelling, 25(6):870{884, 2007.

[10] Richard D. Taylor, Philip J. Jewsbury, and Jonathan W. Essex. A review of protein-small molecule docking methods. Journal of computer-aided molecular design, 16(3):151{166, 2002.

[11] Oleg Trott and Arthur J Olson. Autodock vina: improving the speed and accuracy of docking with a new scoring function, efficient optimization, and multithreading. Journal of computational chemistry, 31(2):455{461, 2010.

[12] PDB File Format - Contents Guide (3.30). World Wide Web, http://www.wwpdb.org/documentation/format33/v3.3.html.

6 A MODEL ARCHITECTURE FOR BIG DATA APPLICATIONS USING RELATIONAL DATABASES

Effective Big Data applications dynamically handle the retrieval of decisioned results based on stored large datasets efficiently. One effective method of requesting decisioned results, or querying, large datasets is the use of SQL and database management systems such as MySQL. But a problem with using relational databases to store huge datasets is the decisioned result retrieval time, which is often slow largely due to poorly written queries / decision requests. This work presents a model to re-architect Big Data applications in order to efficiently present decisioned results: lowering the volume of data being handled by the application itself, and significantly decreasing response wait times while allowing the flexibility and permanence of a standard relational SQL database, supplying optimal user satisfaction in today's Data Analytics world. We experimentally demonstrate the effectiveness of our approach.

6.1 INTRODUCTION

Big Data applications span multiple business contexts, from social media to financial interactions to scientific computing. The incredible volume and continued growth of data [1] and its associated derived knowledge has led to a gap between the amount of data to be reviewed and the ability of a human individual to ‘take it all in’ in order to make a considered decision. Accordingly, companies have addressed this issue by creating applications to automate the process of decisioning: providing graphs, dashboards and useful subsets of the data to the human users that need the data for their day-to-day jobs.

But what is Big Data? The term is thrown around so much in business and scientific contexts, with such varied meanings [2][3][4][5][6], that a quick moment to clarify the meaning

in the context of this paper is not out of place. Big Data is a blanket term for any collection of data sets considered to be too large and complex to easily manage, curate and process using traditional means: i.e. database management tools and/or data processing applications. The main descriptors were defined over 10 years ago as the 3 Vs: Volume, Velocity and Variety [7]. Volume is simple to understand: a lot of data. Big Data sizes are a moving target [8][9], but in most current contexts, it usually includes data sets with sizes from a few terabytes [10][11] to many petabytes [12]. Or more simply put, "Big data is what happened when the cost of storing information became less than the cost of making the decision to throw it away." [13]

Velocity also is fairly self-explanatory: the data accumulates swiftly. Big data is usually composed of collected observations over time, and/or recorded transactional data points, for a distinct set of entities. And this set of entities is usually much smaller than the total number of observations being collected. Most data becomes Big due to repeated entries for the same entities over time and/or space [14]. For example, a retailer might have millions of customers, thousands of inventory items and hundreds of stores, but will log billions of individual transactions annually. Telecommunications traffic entities can include voip devices, cable modems and subscribers, and are limited to the number of physical devices in use, but will log statistical data on a per-minute to per-hour basis for each entity, accruing millions and millions of records daily. Scientific measurements can produce data that is even bigger: the Large Hadron Collider experiments alone generate more than 20 petabytes per year [12].

So what about the last term, Variety? Variety means that the data comes from a lot of different places. Does that mean that there is structured data from different sources involved, such as different systems, different databases, different applications, etc? Yes, very likely. What kind of data is considered structured data? Data that conforms to an explicitly defined data

model is structured data. Though this type of data is often found in a database, this may also be found in predetermined record fields inside a file, or set of files. Does the term "Big Data" mean that there must be semi-structured data involved? No. Can the Variety of data being saved include semi-structured data? Potentially, if the company paying the bills wants to save that kind of data, or wants to use that kind of data in their analysis. What kind of data is considered semi-structured? XML documents, JSON documents, email and EDI are all examples of semi-structured data. (Basically, semi-structured data usually has a defined structure, but it is not data that would normally conform to a relational database structure.) Does Big Data always include unstructured data? No. Can unstructured data be included in the Variety of data being saved? Possibly, if the company paying for the storage of the Big Data and the subsequent analysis of that information wants to save and potentially use that kind of data in their analyses. What kind of data is considered unstructured? Information without a pre-defined organization (structure), and/or without a defined data model that is meaningful to the Content of the data itself, is usually considered to be unstructured. This often includes the textual content of any kind of documents, texts, tweets, instant messaging chats, presentations, emails, etc., but can also include audio, video and image files.

Most companies largely focus on the analysis of structured data, also known as traditional Business Intelligence [15]. Aside from the well-understood benefits of Business Intelligence that has been around for multiple decades, what do we gain by having more and varied data to work with? The ability to save huge amounts of data in traditional databases has allowed unparalleled amounts of traditional trending, forecasting, and pattern detection. People often are oblivious about just how much information is being collected. For example, stores do indeed keep track of all of their customers' purchases over time: that information belongs to the store, and can be

scrutinized for insights into the lives of their customers. The analysis of this data is not a "neutral or unguided process" [16]. A company has to pay for the storage and for the analysis of their data. It is not to be expected that the company would perform these activities without a reason: under normal circumstances, do you often personally drive down a road that does not contain at least one destination that you would like to reach? Driving takes up your time, your money (fuel for the vehicle), and your assets (the car and you): you are most likely to use these things for reasons that will benefit you. A company is made up of individuals: they are no different.

Therefore the largest amounts of money driving the analysis of Big Data will be the same as those that previously drove traditional Business Analysis: sales and marketing [17]. Do you eat out on Wednesdays for lunch? With friends? Do you shop for groceries on the way home from work on Mondays? Do you always buy brand name sodas? Do you only purchase buy-one-get-one-free cereal? Do you tend to buy the same brand of hot dogs, but vary over the type of hot dog buns, depending on what's on sale? How about your clothing and furniture purchases? Do you tend to purchase from big retailers, or boutique or specialty stores? Are you pregnant? Do you even know about it yet? Depending on where you shop, your store may well know about it before you do [18]. Needless to say, with the unparalleled level of data being stored, privacy is a big concern as well [19][20]. This is especially true in its downstream implications. Not only is it unnerving that a retailer knows that you prefer purple pajamas over any other color, and that you always purchase your holiday gifts the day before Christmas, employers and insurance companies have now made a practice of purchasing this historical customer spending information from these retail companies in order to develop employee profiles [21]. Again, currently it is often about sales and marketing: Blue Cross and Blue Shield

of North Carolina stated that their use of the purchased data was to try to "sell you something that can get you healthier" [21]. But scrutinizing this historical customer spending information can give employers other insights that are generally not currently exploited (but could easily be) from the same data set, especially when an employer is researching a new hire: do you consistently buy Late Birthday gifts or cards? Do you consistently purchase alcohol or tobacco products? Or for an existing employee: Are you often shopping on days when you've called in sick? The majority of the insight and advantage from large-scale data analytics will be directed at answers to the questions that the companies have an interest in asking, and these questions are usually tailored to the specific needs of their business.

One of the biggest problems with any kind of traditional Business Intelligence data analysis, is that the data is always filtered. The lowest-level employee working on an assembly line will by definition see more of the day-to-day interaction with the product on the assembly line than the company president. The company president actually has the least granular data about the product at his fingertips: to make critical operations decisions, he must rely on information that has been filtered through multiple systems and rolled-up, summarized reports.

The newer concepts in Business Intelligence include the analysis of semi-structured and unstructured data, both from company-held and public sources. These public sources can include anything posted on the web such as personal webpage pages and blogs, to Twitter tweets, Facebook, Instagram, and Google+ posts, YouTube uploads, and Pinterest pins. The textual content of the unstructured data is not a "silver bullet" or cure-all, but analysis of publicly available social media data and can often be used to provide swift customer service responses

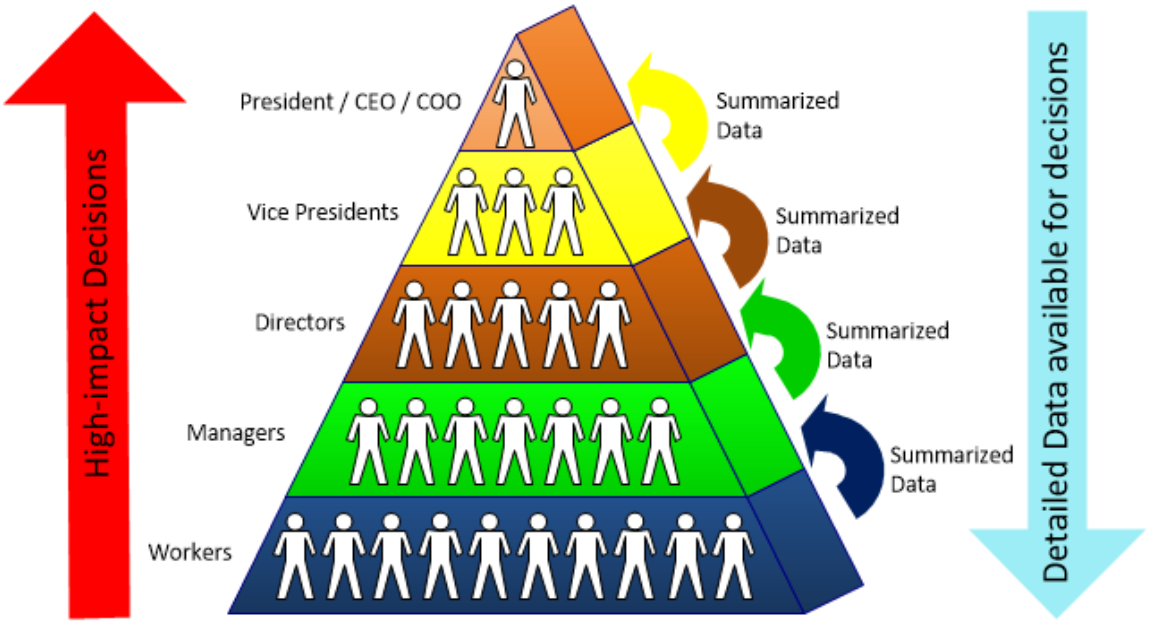


Figure 11 **Traditional Business Intelligence: the people who must make the high-impact decisions tend to have the least granular data available to them for analysis.**

[17], and direct public relations opportunities [22][23] as well as to develop useful predictive models regarding the opinions or sentiments of online responders (anyone online who feels the urge to tweet or post).

After all, how many times do you complain about a company's customer service versus praise it? It is expected that if you purchase a pizza, it arrives on time. It is expected that if you pay for an item, that it performs exactly as expected. If this doesn't happen, then you complain. The current fastest way to a customer service representative's ear is not to pick up the phone, but rather to tweet to the world about how much the company is disappointing you using their company name as the hashtag [24]. No company wants their online name dragged through the ether-mud, and therefore this online whining [25] will in many cases instantly blackmail the company into quickly (and publicly) remediating the "problem" that you encountered [24].

Most unstructured data mining is currently focused on marketing and IT - getting better

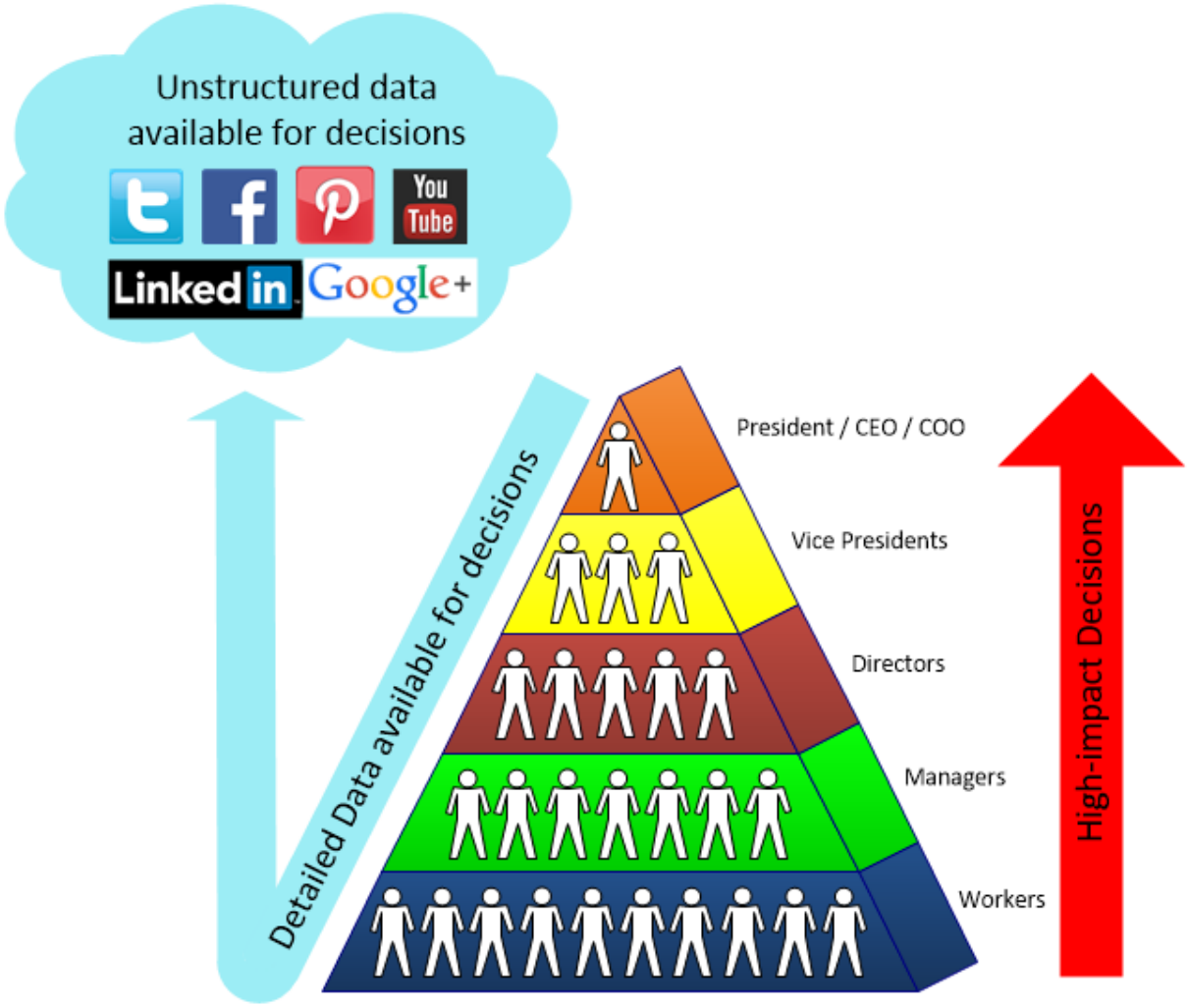


Figure 12 **Analysis of semi-structured and unstructured data can provide direct and detailed information about what current and potential customers are interested in right now.**

information about what current and potential customers are interested in right now, early detection of customer problems with a particular product (or a competitor's product), new marketing stratagems and business practices, price changes, etc. But due to the prevailing public sentiment regarding the freedom to vent online, the general public now voluntarily gives up a great deal of additional information – even categorizing it with hashtags for easier machine searching and automated processing - and sometimes human resources can also gain insight into

reasons for employee turnover, or public relations and marketing can take advantage of a power outage during the super bowl to turn a costly advertising loss into a Social Media win [22].

But while mining unstructured data can allow a company to get customer feedback data directly from "the horse's mouth" as the old saying goes, caution and good sense are still required. Using search and indexing technology to monitor publicly-available unstructured data is a useful strategy, but it doesn't always yield usable results [26][27]. An example would be the 2011 big data project that analyzed tweets and other social media outlets to attempt to predict the U.S. unemployment rate via sentiment analysis, by month, based on word counts in these unstructured data sources for words such as jobs, unemployment and classifieds [26][27]. In their excitement over finding an upswing in the sentiment analysis word count in October 2011, researchers didn't notice that their word choice was fouled: Steve Jobs had died, and it had lit up the Twitter feeds, regardless of the employment or unemployment of the tweeters. Not only that, but many sources now use your online Friends on Facebook, and LinkedIn to determine your "character and capacity" for loans and employment opportunities [28]. In the USA, credit agencies commonly are required to drop information off an individual's credit report after 7 years (10 years for a bankruptcy proceeding), but information listed on Google, Twitter, and Facebook never dies.

The biggest problem in dealing with Big Data is not storing the data itself, but is deciding what data is useful for a particular decision request. Different types of data lend themselves to different analysis techniques. Most traditional Business Intelligence employs descriptive analysis techniques to create reports including statistical functions (max, min, mean, etc.) to describe the population of data that is being reviewed. The reports often include summary data tables, graphics (charts), and explanatory text. This sort of quantitative analysis is typically used

with structured data. With inductive analysis, you begin by examining concrete examples of your data. You then apply qualitative methods to the data population in order to understand the domain and determine what the important factors are. This works especially well for unstructured data where you may not know what insights to expect from your data population. [29]

The most popular querying language is SQL (standard querying language). Support for SQL is built in to all modern database management systems, and business-analyst resources who know how to both understand the business drivers as well as how to use the querying language are relatively easy to hire. Referential databases are the easiest method for current IT business analysts to hit the ground running and dig in for answers to common questions such as:

- What happened?
- Why did it happen?
- What will happen next?
- Will it happen again?

But with the huge volume and variety of data involved, naive or untuned queries can take staggeringly long amounts of time to return results, rendering the Big Data application useless.

The biggest issue for Big Data applications is a matter of asking the Right Question. The most famous Big Data question was posed in the fictional work The Hitchhikers Guide to the Galaxy "The answer to the great question of life, the universe and everything." It took seven and a half million years to solve by the computer Deep Thought ...and according to the fictional work, it was finally calculated: forty-two [30]. Which, obviously, is a farcically useless datum. This highlights one of today's most obvious issues with the Big Data environment: the problem isn't putting the data into the database, the problem is getting useful data back out [14].

Previous articles explicitly depicted difficulties of storing all of the data from a very large relational dataset in memory [14] in order to return it to a calling application. But the fact remains that the calling application does not need all of the data. You do not need to hold every dollar bill from your bank account in your hands in order to pay the bill at your favorite coffee shop. You only need to have in your hand the amount required to perform the bill-paying activity. By this same argument, in order to answer many Big Data decisioning questions, all of the data points from the stored Big Data do not need to be returned to the calling application: only enough to answer the question being asked.

The calling application needs a question answered. Using the database as just a receptacle to hold data sets, while the calling application does all of the work, is a misuse of available resources and a waste of bandwidth. Imperative database programming elements including stored procedures, functions and triggers that sit directly in the database and do not incur the overhead of network traffic, and have direct access to the data itself, allow the preponderance of the heavy-lifting data analysis work to be performed at the database layer and are supplied by every major database vendor [31][32][33][34][35][36]. In most cases this is faster than the well-known map-reduce paradigm [37]. That's largely because the basic idea for this framework has existed in parallel SQL database management systems for over 20 years [37].

6.2 A NEW MODEL

"I checked it very thoroughly," said the computer, "and that quite definitely is the answer. I think the problem, to be quite honest with you, is that you've never actually known what the question is"
-- Deep Thought, The Hitchhikers Guide to the Galaxy[30].

6.2.1 Considerations

The requirements for any given piece of software can be divided into two basic categories: functional and nonfunctional. Functional requirements are descriptions of what the application should do: services provided, how the application should react to inputs, and how it should behave in particular scenarios [38]. They depend strongly on the type of software being developed, the expected users/audience, and the type of environment where the software is expected to be used. Nonfunctional requirements are constraints on the application's services or functions: timing constraints, security restrictions, storage requirements, standards, etc. These often apply to the application as a whole rather than to any one individual features or service in particular [38]. The problem is that nonfunctional requirements may be more critical than functional requirements. If they are not met, then the application may be useless to its expected audience.

6.2.2 Good Software

There are three basic things that every piece of software should be:

1. Usable
2. Dependable
3. Maintainable

Number one in the list relates to the functional requirements, and numbers two and three contain the nonfunctional requirements. But in in examining the reasons why a customer will choose to not use a piece of software, especially a web application, frustration over lag (long wait times between when an application request is sent and when the application responds) far outweighs the benefits of all the bells and whistles. Though this especially comes into play with secured web applications, where slow application responses can have additional cascaded effects

[39] up to and including timing out the customer from the application entirely. Just because a tool is processing a ton of data, does not mean that the end-user will give any consideration to the volume and breadth of the decisioning requirements. The extreme frustration generated from the inability of the software to meet the nonfunctional requirement of a timing constraint can engender such a bad feeling towards an application (almost a hatred) that customers will unreservedly lambaste it for taking even a few seconds longer to respond than expected: a quick google search of the words "I hate lag" yields around 14 million results, for different software, websites and devices. This is not a small concern for a human-facing application.

6.2.3 Back to Basics: Engineering Principles

> *"Other Engineering disciplines have principles based on the laws of physics, biology, chemistry or mathematics... Because the product of software engineering is not physical, physical laws do not form a suitable foundation. Instead, software engineering has had to evolve its principles based solely on observations of thousands of projects."*
> *-- Alan Davis, IEEE, November 1994 [40]*

Software principles have been derived from the collected wisdom people who learned through experience. They will continue to evolve as the discipline grows. Experience is gained by practice: as more experience is gained, the collected wisdom grows and new principles evolve from the massed experience.

Davis's principles are a tried and true checklist of back-to-basics items that any application refactoring effort should make the effort to walk through. The first item to address is Davis's principle number 10, "Get it Right before you make it Faster [40]." Though of course increased speed is always a goal, it was a mandatory functional requirement that the case study Big Data application return accurate decisioned results. Davis's principle number five, "Evaluate Design Alternatives [40]," discusses the need to examine a variety of application

architectures and algorithms after the requirements have been agreed upon. In the case study, that was definitely something that would need to be examined. Davis's principle number seven, "Use Different languages for Different phases [40]," is incredibly clear about pointing out that using object orientation for all phases, when it is not what is optimal for a particular phase, is useless. There's an old saying that states, 'When all you have is a Hammer, then every Problem looks like a Nail.' The problem lies in that the drive to find simple solutions to complex problems, which often blinds developers to solutions outside their immediate comfort zone, forcing draconian non-functional requirements such as always using ORM, or only using Java (or any other single language) as part of the development standards, etc., instead of determining where each portion of the solution would be optimally built. Making everything use the same language does not make the solution simpler. In many cases, it can make a solution much harder. What is simple to accomplish in one language may be incredibly difficult to accomplish in another.

In the original version of the case study software, a Big Data web application to interpret cable modem port analytics, the application layer encapsulated all of the business logic. The Big Data application used a naive inline SQL query to the database, which would then send all of the requested data back to the application, which would then manipulate the data to provide a decisioned result: essentially doing the processing work in two places. This process was expensive and painful, as the I/O to repeatedly export the full data set from the database and then perform decisioning in the web application layer was limited by both the processor speed and the cache read/write speed (not to mention the network speed itself, depending on the end-user's location) – none of which can be controlled by the application developers. This strategy was difficult to maintain, as any changes to the business logic would require a change to the

application layer; and at runtime the complex, weighty queries were sent to the database for processing by end-users on-demand, sometimes causing lengthy waits as the Big Data application waited for its results to be returned from the database for further processing, especially with the larger datasets. Here also was an area that needed to be examined.

6.2.4 Breaking it Down: Examining The Alternatives

More than 20 years ago, support for procedural elements was wrapped into the SQL Standards [41]. In 1996, procedural elements, views, and metadata elements were officially added to SQL (as an extension of SQL-92)[42]. As perspective, Sun released the first public implementation of Java in 1995 [43]. In 1999, procedural elements were so far incorporated into standard DBMS implementations, that database vendors were touting their merits at conferences worldwide [44][45]. All major database vendors support stored procedures [31][32][10][34][35][36].

So what are these procedural elements? Stored routines (procedures, functions and triggers) are chunks of code stored directly in the database. They can use standard coding constructs such as IF, CASE, LOOP, WHILE, etc., and they can accept and return parameters, including returning result sets. SQL is an imperative language, i.e. it is inherently canted towards a procedural programming paradigm; and unlike the declarative model of Object-Oriented languages, a procedural programming paradigm allows the programmer to specify an exact step-by-step process to solve a particular need (or problem). These procedural elements in SQL are especially useful for culling extensive, expensive, and/or complex processing of data out of the application layer; and simplifying application logic by then allowing the application to call a single stored procedure instead of multiple inline queries. Procedural elements, stored within the database itself, avoid the overhead of network traffic altogether as they are stored

within the database engine and therefore have direct access to the data to be manipulated. As the syntax is already checked when the stored routine is compiled into the database, there is a lesser likelihood of data corruption via application misusage of dynamically generated inline SQL queries. With stored procedures acting as an API, the application does not have to know the details of the relational database implementation, smoothing over the Object-Oriented application complaint of Object-Relational Impedance Mismatch. Again going back to Davis's rule "Use Different Languages for Different phases [40]," by breaking out the data-manipulation portion of the algorithm, it is no longer necessary to force a nonfunctional requirement of object-oriented design for all layers of the application. Davis's principle number 23, "Encapsulate [40]," also lends itself to this approach. As stated in his article, it is a simple concept that results in software that is easier to test, use and maintain.

6.3 EXPERIMENT

6.3.1 Solution Viability

The purpose of the experiments was to determine whether the proposed algorithm could be implemented to take advantage of a database procedural elements, in order to facilitate decisioning larger data sets. Three (3) iterations of the request for a decisioned result set were used for the purpose of the experiment. The first was serviced by a generic, straightforward database SQL query submitted by the development team; the second was serviced by an optimized version of the same generic, straight-forward database SQL query; and third was serviced by the proposed model. Comparisons were created using the volume of data required to complete the database transaction, the time required to run the database transaction, and the number of records returned to the calling application.

6.3.2 Dataset

In a Big Data environment, the largest problem is not obtaining the data or even storing it, but rather getting useful answers derived from the raw data within a reasonable timeframe. One application requiring said swift, useful answers is deriving port analytics for the modems on a cable modem termination system (CMTS). A CMTS is the piece of equipment that a cable company uses to connect each of their headends to the Internet on one side, and their subscribers on the other. In order to provide continuous high-speed data services to its subscribers, a cable company needs to monitor not only the reported health of the CMTS itself, but also the health of each modem in a subscriber location. The values of multiple data points can be used to encapsulate the health of a typical modem, including signal noise ratio, T3 and T4 timeouts and code word errors [46]. These values can be further rolled up to give a measure of the health of the ports on the CMTS itself.

For the case study experiment, to create their Big Data application, a survey was conducted to determine what database management system (DBMS) would provide the best performance and maintainability for the size of data (around 10 TB for a rolling 6 month period) to be preserved, and alternatives (including both SQL and NoSQL) were examined. MongoDB, specifically, was evaluated closely over several months, but was both unable to keep up with the write and throughput load, and was very difficult for end-users to manipulate afterwards in order to analyze the captured data (getting useful answers). The relational DBMS, Oracle MySQL, was eventually chosen for throughput speed and ease of use.

Internal industry experts had defined the evaluation criteria for each of the 9 attributes to be evaluated: ranges were defined for Red/Bad, Yellow/Warning, and Green/Good. Raw data

was collected for each modem on a periodic basis between 15 and 60 minutes. The raw data was partitioned into daily segments, over multiple relational tables.

To make useful decisions about the collected data, it was required that a current measurements snapshot be created against the evaluation criteria. The twist was that each attribute had to be evaluated on a basis of net difference between the most recently collected data and the next most-recent valid set of data collected at least 60 minutes previously.

6.3.3 Results

6.3.3.1 Iteration 1: Naive, inline query

Given a defined set of ranges, a straightforward method to calculate the set of records to be displayed would be to take the set full set of records for the current collection and join them with the full set of records from the previous collection, determining the net difference between the attributes of the two sets in order to perform classification range evaluation (Red, Yellow, Green).

```
for all records in set
for each attribute to be graded
     case (previous – current)
          if value in range_1 then RED
          if value in range_2 then YELLOW
          if value in range_3 then GREEN
```

When the database was initially created, this naïve method gave a reasonable response time (<1s), but as the volume of collected data grew to its current size, the naïve method understandably took longer and longer to obtain results. Using this naïve method as an inline

query in the Big Data application, approximately 187G of raw data would be processed to obtain a calculation each time an individual

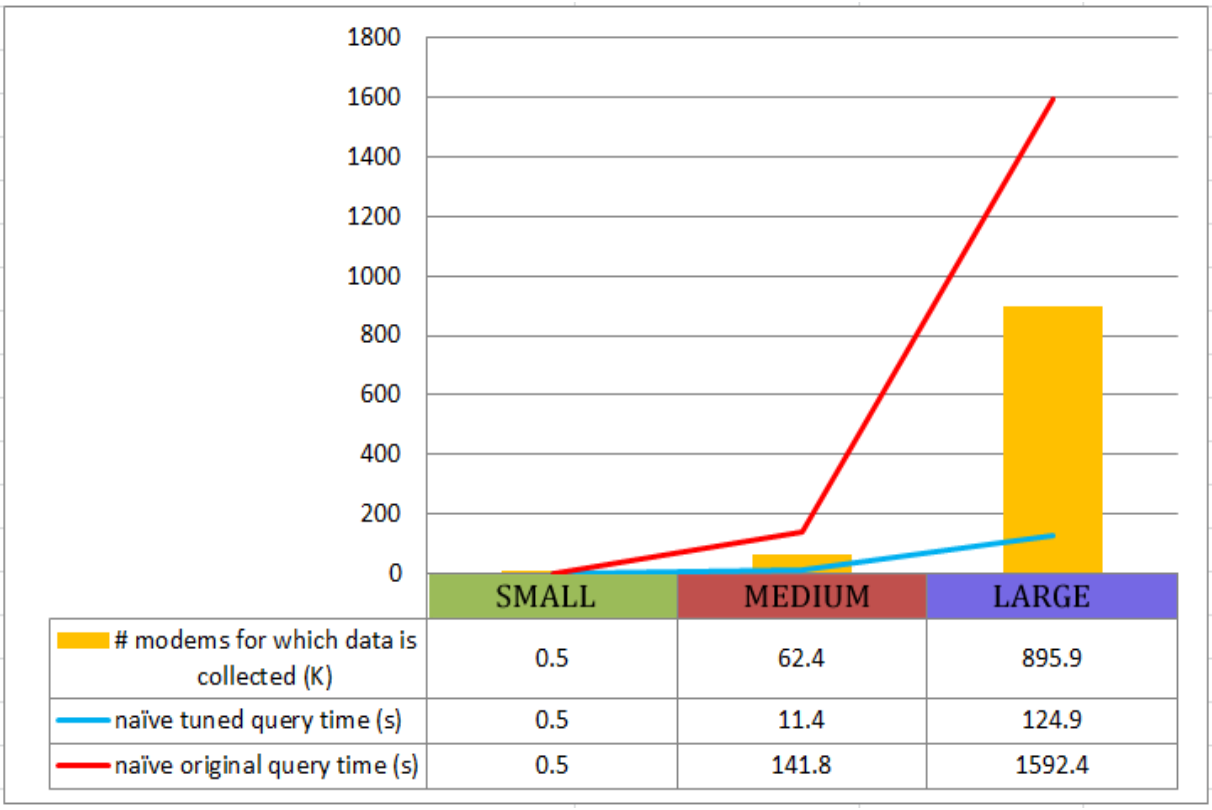


Figure 13 Naïve Tuned (Optimized, inline query) versus Naïve Original (Naive, inline query) Timing Results.

report was run. At the peak volume record set during the testing period, the naïve method averaged 26.5 minutes to process through the approximately 300 million raw relational records gathered hourly into a <100,000 record "snapshot" evaluation response.

6.3.3.2 Iteration 2: Optimized, inline query

Tuning the naïve query to prune down the decision tree for the result set did help significantly, lowering the response time by more than a factor of ten, but for larger customers this still meant that each individual report did not return data for over 120 seconds – more than 2

minutes – which is an unacceptably high wait time (lag). Clearly, a different approach was required.

6.3.3.3 Iteration 3: Proposed model

As suggested in section II.D, it was proposed that in order to optimize the query response time for customers, that the complex queries be culled out of the application layer, and constructed via a stored procedure into an indexable temporary relation / materialized view. Refreshed periodically, faster and simpler queries could then be drawn out of the materialized aggregate view to populate the graphical reports for calling Big Data application.

6.3.3.4 Algorithm:

a) Create a shell structure that encompasses all required columns, including
    i) current values
    ii) previous values
    iii) net values
    iv) other identifying information for required joins
b) Insert the set of devices that meet the business requirements
c) Retrieve the current data points from raw data and update the appropriate fields
d) Retrieve the previous data points from raw data and update the appropriate fields
e) Calculate the net differences and update the appropriate fields
f) Apply the business logic ranges (using the net differences) to evaluate each record

So that there would never be a situation where the cable companies did not receive "live data" on their graphical snapshot reports, the data from the previous evaluation would remain

(prominently displaying its timestamp) available for review until the latest data calculations were fully complete.

6.3.3.5 Performance Analysis:

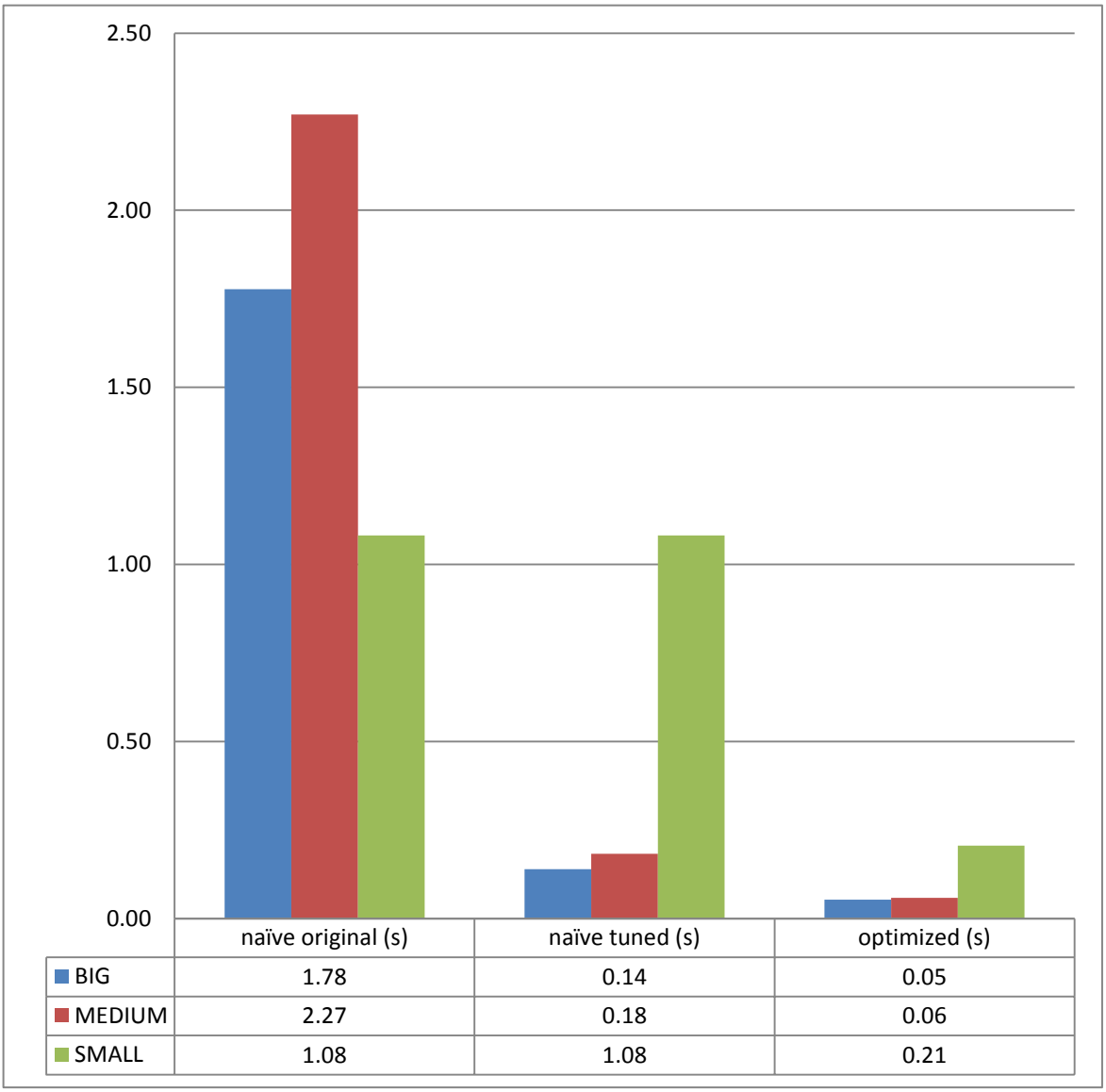
As the previous and current raw data to be evaluated for the net comparisons is now selected in different steps, this process immediately halves the amount of data being processed in memory at any one time as compared to the naïve algorithm.

	SMALL	MEDIUM	LARGE
number of modems for which data was collected(K)	0.5	62.4	895.9
naïve query (raw file size in G)	0.3	28.1	187.3
optimized query (raw file size in G)	0.1	14.0	93.6

Figure 14 Data set characteristics.

The larger the raw data became, the more evident it became that the materialized view optimization process was much more efficient. The application was able to obtain the required swift, useful answers regarding cable modem port data analytics from the millions of records of raw data within a reasonable timeframe using the outlined method. By making use of the indexable temporary relation / materialized view, the query-retrieval time was cut down 33x. This technique is especially useful in a scenario when calculations can be reused over a period (in the case study, 15-60 minutes), as the results are cached in the indexable materialized view and are thus available for swift retrieval without necessitating the overhead of recalculation.

Dividing the total query processing time by the number of modems, one can see that while a benefit is achieved by using the algorithm even in cases of smaller datasets, the largest benefit is felt when addressing bigger datasets.



	naïve original (s)	naïve tuned (s)	optimized (s)
BIG	1.78	0.14	0.05
MEDIUM	2.27	0.18	0.06
SMALL	1.08	1.08	0.21

Figure 15 Query cost per datum performance comparison.

6.4 CONCLUSIONS AND FUTURE WORK

Big Data applications cover multiple contexts, providing valuable business insights into very large data sets. Big Data usually is defined to mean data sets with sizes beyond the ability of commonly used software tools to capture, curate, manage, and process within a tolerable elapsed time. Big data sizes are a constantly moving target: as of 2012 these sizes range from

hundreds of gigabytes to a few dozen terabytes to several petabytes in a single data set. The largest Big Data datasets can range to many petabytes in a single dataset, but according to the 2012 IOUG Big Data Strategies Survey sponsored by Oracle [47] almost 90% of respondents reported that that the total amount of data managed (all sources) was under 1 petabyte. Of that data, over three quarters (77%) of the respondents considered structured data (transactional database data) to be most important to their business.

From this information, one can infer that the majority of Big Data analysis currently occurring in private industry is largely dealing with structured datasets sized in the multi-terabyte range. In many cases, the volume of data is hard to handle not because the database engine is incapable of handling it, but because either the data model or the application queries were ill-designed: the developers thought only of how the application would be putting the data INTO the database, and not how the data would be coming back OUT [14].

Effective Big Data applications must handle the retrieval of decisioned results based on stored large datasets efficiently. One effective method of requesting decisioned results, or querying, large datasets is the use of SQL and database management systems such as MySQL. But a perceived problem with using relational databases to store huge datasets is the decisioned result retrieval time. Relational databases are often seen as slow by developers because of poorly written Big Data application queries / decision requests. But the business analysts that work with the data daily to perform ongoing discovery and analysis prefer relational databases because they are easy to understand and easy to query using SQL and tools that the analysts already know how to use. This work presented a model to re-architect Big Data applications in order to efficiently present decisioned results: lowering the volume of data being handled by the application itself, and significantly decreasing response wait times while allowing the flexibility and permanence

of a standard relational SQL database, supplying optimal user satisfaction in today's Data Analytics world.

Using Davis's software engineering principles, we developed a strong model to re-architect Big Data applications in order to efficiently present decisioned results. By taking advantage of the procedural elements in SQL and culling extensive, expensive, and complex data processing out of the application layer, we simplify the application logic by modifying the Big Data application to use stored procedures instead of calling multiple inline queries. Stored within the database itself, these procedural elements not only avoid the overhead of network traffic altogether, but also have direct access to the data to be manipulated. With stored procedures acting as an API, the application does not have to know the details of the relational database implementation: the syntax already having been checked when the stored routine is compiled into the database, there is a lesser likelihood of data corruption via application misusage of dynamically generated inline SQL queries.

We experimentally demonstrated the effectiveness of our approach using a 10TB cable modem port data analytics dataset. The Big Data application in the case study (cable modem port data analytics) is still under development. As the outlined Big Data application model already works comfortably with terabyte-sized data sets, future work will include optimized aggregate reporting and trending over time.

6.5 ACKNOWLEDGMENT

This work was supported in part by the Georgia Cancer Coalition (RWH is a Georgia Cancer Scholar) and the Georgia State University Molecular Basis of Disease Initiative.

EEAAD gratefully thanks Momentum Telecomm for the opportunity to work with the cable modem quality of service (QoS) DOCSIS classification data.

6.6 REFERENCES

[1] Gantz, J., Reinsel, D. "Extracting Value from Chaos," The 2011 IDC Digital Universe study sponsored by EMC, June 2011.

[2] Rouse, Margaret. "What Is Big Data?" Big Data and Cloud Business Intelligence. TechTarget, 30 June 2014. Web. 10 July 2014.

[3] Big Data / Analytics / Strategy / FP&A / S&OP / Strategic Planning / Business Analytics / Innovation. "What is the difference between Busiiness Intelligence and Big Data." LinkedIn [Group page]. Web. 31 January 2013. Retrieved July 01. 2014, from http://www.linkedin.com/groups/What-is-difference-between-Busiiness-1814785.S.210009923

[4] Davis, Jim. "What Kind of Big Data Problem Do You Have?" Web log post. What Kind of Big Data Problem Do You Have? SAS Institute, Inc., 08 Oct. 2012. Web. 02 Mar. 2014.

[5] Payandeh, Fari. "BI vs. Big Data vs. Data Analytics By Example." Foreground Analytics Big Data Studio. N.p., 24 Aug. 2013. Web. 04 Mar. 2014.

[6] "Definitions of Big Data." Definitions of Big Data | Opentracker - Digital Analytics. Opentracker, n.d. Web. 01 July 2014.

[7] Laney, Douglas. "3D Data Management: Controlling Data Volume, Velocity and Variety". Gartner. Retrieved 6 February 2001.

[8] Johnston, Leslie. "Data Is the New Black." Web log post. The Signal: Digital Preservation. The Library of Congress, 14 Oct. 2011. Web. 04 June 2014.

[9] "Community Cleverness Required." Editorial. Nature 4 Sept. 2008: n. pag. Community Cleverness Required. Nature Publishing Group, 3 Aug. 2008. Web. 06 June 2014.

[10] IBM. New IBM Big Data Technology for Dramatically Faster Data Analysis and Decision-Making Enters the Market. New IBM Big Data Technology Enables Faster Data Analysis. IBM News Room, 26 June 2013. Web. 04 Mar. 2014.

[11] "Utilizing Better and Faster Intelligence to Improve Strategic Decision Making: Companhia De Seguros Tranquilidade." Pivotal Software. Pivotal, Jan. 2014. Web. 04 Mar. 2014.

[12] CERN. N.p.: CERN, n.d. Accelerating Science and Innovation - CERN Document Server. CERN, 30 May 2013. Web. 01 July 2014. <http://cds.cern.ch/record/1551933/files/Strategy_Report_LR.pdf>.

[13] O'Reilly, Tim (timoreilly). "'Big data is what happened when the cost of storing information became less than the cost of making the decision to throw it away.' - George Dyson #longnow" 19 March 2013, 8:51PM.

[14] Jacobs, A. "The Pathologies of Big Data," ACM Queue: Tomorrow's Computing Today. vol. 7, issue 6, pp. 1-10, July 2009.

[15] Thurai, Andy, and Basu, Atanu. "Prescriptive Analytics: An Adaptive Crystal Ball." Tech News and Analysis. Gigaom, 22 Mar. 2014. Web. 01 July 2014.

[16] Stanley, Jay. "Big Data and Big Money." Web log post. Big Data and Big Money | American Civil Liberties Union. American Civil Liberties Union, 10 July 2014. Web.

[17] Bhatia, Akash. "Social Business The Rise of Unstructured Data." Infosys - Social Media Business Impact | Structured and Unstructured Data. Infosys Limited, May 2011. Web. 01 July 2014.

[18] Duhigg, Charles. "How Companies Learn Your Secrets." The New York Times. The New York Times, 18 Feb. 2012. Web. 01 July 2014.

[19] Terdiman, Danel. "At CNET's SXSW 'big Data' Panel, Sparks Fly over Privacy." At CNET's SXSW 'big Data' Panel, Sparks Fly over Privacy - CNET. CNET, 12 Mar. 2012. Web. 01 July 2014.

[20] Stanley, Jay. "Eight Problems With "Big Data"" American Civil Liberties Union. American Civil Liberties Union, 25 Apr. 2012. Web. 01 July 2014.

[21] Wieczner, Jen. "How the Insurer Knows You Just Stocked Up on Ice Cream and Beer." The Wall Street Journal. Dow Jones & Company, 25 Feb. 2013. Web. 01 July 2014.

[22] Fung, Katherine. "Oreo's Super Bowl Tweet: 'You Can Still Dunk In The Dark'" The Huffington Post. TheHuffingtonPost.com, 04 Feb. 2013. Web. 04 June 2014.

[23] Watercutter, Angela. "How Oreo Won the Marketing Super Bowl With a Timely Blackout Ad on Twitter | Underwire | WIRED." Wired.com. Conde Nast Digital, 02 Feb. 2013. Web. 01 July 2014.

[24] Palmquist, Matt. "How to Order a Social Media Pizza with Extra Text Mining." Web log post. Strategy+business. PwC Strategy& Inc., 25 July 2013. Web. 01 July 2014.

[25] Karen Merrell Puntney (Karen Merrell Puntney). "except when it is for later posting on the restaurant's FB page as a complaint (seeeeeeeee how horrible!) to get free stuff" 14 July 2014, 1:02PM. https://www.facebook.com/ifyoucantaffordtotip/posts/10152209242393091?reply_comment_id=10152210123308091&total_comments=1

[26] Arellano, Nestor E. "Why Big Data Is Not Always Good Data." IT World Canada. IT World Canada, 8 Sept. 2013. Web. 01 July 2014.

[27] Tucci, Linda. "Big Data Can Mean Bad Analytics, Says Harvard Professor." Big Data Can Mean Bad Analytics, Says Harvard Professor. TechTarget, July 2013. Web. 01 July 2014.

[28] "Stat Oil." The Economist [London, England] 9 Feb. 2013. The Economist. The Economist Newspaper Limited, 09 Feb. 2013. Web. 01 July 2014.

[29] "Big Data car Low-Density Data? La Faible Densité en Information comme Facteur Discriminant." Lesechos.fr. Les Echos News, 03 Apr. 2013. Web. 01 July 2014.

[30] Adams, D. (1979) The Hitchhikers Guide to the Galaxy. Germany: Pan Books.

[31] PostgreSQL: Documentation: 9.3: Procedural Languages. PostgreSQL: The world's most advanced open source database. Retrieved March 1, 2014, from http://www.postgresql.org/docs/current/interactive/xplang.html

[32] MySQL 5.7 FAQ: Stored Procedures and Functions. MySQL Reference Manual. Retrieved March 2, 2014, from http://dev.mysql.com/doc/refman/5.5/en/faqs-stored-procs.html

[33] CALL statement. IBM iSeries Information Center, Version 5 Release 3. Retrieved March 2, 2014, from http://publib.boulder.ibm.com/infocenter/iseries/v5r3/index.jsp?topic=/db2/rbafzmstcallstmt.htm

[34] "Stored procedure." Sysbase Wiki. Retrieved March 1, 2014, from http://www.petersap.nl/SybaseWiki/index.php/Stored_procedure

[35] "Stored Procedures (Database Engine)." Microsoft Technet: SQL Server. Retrieved March 1, 2014, from http://technet.microsoft.com/en-us/library/ms190782.aspx

[36] Stored Procedures and Functions. Oracle Documentation. Retrieved March 1, 2014, from http://docs.oracle.com/cd/B28359_01/server.111/b28318/data_access.htm#CNCPT1776

[37] Pavlo, Andrew, Erik Paulson, Alexander Rasin, Daniel J. Abadi, David J. DeWitt, Samuel Madden, and Michael Stonebraker. "A Comparison of Approaches to Large-scale Data Analysis." Proceedings of the 2009 ACM SIGMOD International Conference on Management of Data (SIGMOD '09) (2009): 165-78.

[38] Sommerville, I. (2011) Software Engineering, 9th Edition. Boston, Massachusetts: Pearson Education, Inc.

[39] United States Department of Health and Human Services (2013, Oct 20). Doing Better: Making Improvements to HealthCare.gov. [Web log post]. Retrieved Mar 02, 2014, from http://www.hhs.gov/digitalstrategy/blog/2013/10/making-healthcare-gov-better.html.

[40] Davis, A. "Fifteen Principles of Software Engineering." IEEE Software 11.6 (1994): 94+.

[41] ISO/IEC 9075:1992 - Information technology -- Database languages -- SQL. ISO - International Organization for Standardization. Retrieved March 2, 2014, from http://www.iso.org/iso/iso_catalogue/catalogue_ics/catalogue_detail_ics.htm?csnumber=16663

[42] Eisenberg, A. (1996). "New standard for stored procedures in SQL". ACM SIGMOD Record 25 (4): 81–88.

[43] "The History of Java Technology". Oracle. Retrieved Mar 02, 2014, from http://www.oracle.com/technetwork/java/javase/overview/javahistory-index-198355.html.

[44] "Guide to SQL Programming: SQL:1999 and Oracle Rdb V7.1." Oracle. Retrieved Mar 02, 2014, from http://www.oracle.com/technetwork/database/database-technologies/rdb/0307-sql1999-130211.pdf

[45] SQL99, SQL/MM, and SQLJ: An Overview of the SQL Standards. Databases and Information Systems - Institute of Computer Science - University of Innsbruck. Retrieved Mar 02, 2014, from http://dbis-informatik.uibk.ac.at/files/ext/lehre/ss11/vo-ndbm/lit/ORel-SQL1999-IBM-Nelson-Mattos.pdf

[46] Data Over Cable Service Interface Specifications. Cable Labs. Retrieved Jan 02, 2014, from http://www.cablelabs.com/specifications/CM-SP-PHYv3.0-I08-090121.pdf

[47] McKendrick, Joseph. Big Data, Big Challenges, Big Opportunities: 2012 IOUG Big Data Strategies Survey. Rep. Oracle, Inc, Sept. 2012. Web. 01 July 2014.

7 CONCLUSIONS

Although most human reasoning is approximate rather than precise in nature, traditional logical systems focus almost exclusively on those modes of reasoning which lend themselves to precise formalization. In recent years, however, in our attempt to design systems which are capable of performing tasks requiring a high level of cognitive skill, it has become increasingly clear that in order to attain this goal we need logical systems which can deal with knowledge that is imprecise, incomplete or not totally reliable.
Lofti Zadeh, ACM Proceedings of the 1986 Winter Simulation Conference, 1986[27]

It is extremely difficult to learn the English language as a human being, much less as a computer. Far harder than mere language is understanding. And coding "gut feelings" into business logic is nightmarishly difficult. But one thing that we can do is approximate an initial prediction, and incorporate (or learn from) those results when one gets it right.

Decisioning, or the machine emulation of human learning and classification, is a nebulous area in computer science. Classic decisioning problems can be solved given enough time and computational power, but discrete algorithms cannot easily solve fuzzy problems. Fuzzy decisioning can resolve more real-world fuzzy problems, but existing algorithms are often slow, cumbersome and unable to give responses within a reasonable timeframe to anything other than predetermined, smaller dataset problems. As the volume of data available for analysis grows in the modern world [1], it is becoming more and more imperative that effective machine-learning solutions are examined that can efficiently handle large datasets.

The optimization of fuzzy decisioning algorithms in order to approximate expert human judgment and disambiguate classifications is incredibly important for working towards the ability to efficiently work with Big Data datasets. For our initial goal of improving on the FDT [6] tool, fuzzy sets were coupled with the Quinlan ID3 partitioning algorithm to generate the decision trees are the basis of the tool's logic [2]. As decision trees are notoriously sensitive to

small changes in training data [3], and largely unable to cope well with uncertain/variable data, FDT 1.0's implementation of fuzzy sets and fuzzy reasoning used approximation to deal with the data set noise: uncertainty/inexact data and fluctuations in data precision, etc. The result was to create a rigorous and effective decisioning tool [6], that was limited in the decisioning power it could exercise over larger sets of data.

The FDT classification algorithm outperforms other machine learning algorithms including Random Forest, C4.5, SVM and Knn [6], and is therefore a prime candidate for integration with a database to facilitate Big Data analysis. In our experiments, we created FDT 2.0, and captured the base data, the fuzzification model, and the decision information into a relational database, from which future decisions can be extrapolated. Large biological datasets from HIV [25], microRNAs and sRNAs [7] were used to measure the effectiveness of the new tool. Even larger experimental datasets included multivariate classifications [9] from a telecommunications case study. Future directions include increasing the accuracy of predictions, decreasing the speed of predictions, incorporating more "learning" elements and working towards decisioning models that can comfortably work with terabyte-sized data sets in a reasonable time frame.

An additional tool that was developed while working on this research was MPDA 1.0, in which we developed a novel approach to determine docking locations using Fuzzy Logic and Shape Determination, by storing the shapes and charges into a relational database. Future analysis and mining of this database is expected to have a great impact as the algorithm is further tuned and the dataset in the database grows to truly large proportions and can be mined for further insights.

The goal of creating something USEFUL as well as technologically superior was maintained and achieved. Not only are the tools themselves useful in producing immediate results, but the created artifacts (databases) are useful as well. The expected audience (end-users) of both tools can independently explore the data afterwards via adhoc SQL querying in the resultant relational databases, drawing further conclusions from the resultant data, and/or manually pruning data as desired from the training sets.

In addition, while working on methods to integrate Fuzzy Decisioning Models with Relational Database Constructs, we created a methodology for the optimization of Relational Database Usage involving large datasets, and in turn, applied this successfully to Big Data. From this we further developed a Model Architecture for Big Data Applications using Relational Databases.

7.1 FUTURE DIRECTIONS

The fuzzy decision tree induction tool (FDT) is still under development. Future directions include increasing the accuracy of predictions, decreasing the speed of predictions, incorporating more automated “learning” elements and working towards decisioning models that can comfortably work with terabyte-sized data sets in a reasonable time frame.

The molecular protein docking algorithm tool (MPDA) is in active development. Current directions for the MPDA tool will include refining the algorithm to include protein-protein shape definitions: Amino acids are the building blocks of proteins and they are also comprised of atoms. The center of the amino acid is the alpha carbon, which would be the new reference point. Because each amino acid has an electric potential, the next version of our algorithm will be tasked with focusing around the coordinate values of the alpha carbon of each amino acid instead of each atom. The current version of the algorithm uses a single probe per shape, with

comparison measures for shape similarity to consider the width, height and depth of a potential docking location. It produces good results in terms of speed of return, running an average of about 4 seconds per protein when searching through all viable docking locations on each protein (Only 27.644 seconds total on the Darunavir molecule against the all of the shapes that correlated to the each of the 7 proteins in the database). Current thoughts on future work include developing a measure of suitability that could be created by determining the alignment of the probes to see if the differences in the shape's probe-length line-up well with each other, potentially factoring in hydrophobicity in binding affinity, and improvements in the determining the actual definition inside of the pocket (the potential docking location) of the shape. In the future, we would also like to adjust the program to account for protein to protein interactions; though this should be a small step as we are already storing the pockets of all proteins we are using, we would only need to also store "peak" information or possible binding regions that could bind to other protein's pockets. Natural usage and growth in our database will cause it to become a repository of similar data: in the future it would also be desirable to use the FDT 2.0 tool against the MPDA dataset to find functional groups of proteins using shared shape information, as geometry plays a large role in protein function, and the determination and classification of functional groups of proteins is a large open area in the bioinformatics community.

While working on methods to integrate Fuzzy Decisioning Models with Relational Database Constructs, we discovered that data retrieval could be abysmally slow in naïve queries on bigger values of **n**. To alleviate this issue, we created "divide and conquer" optimization strategies for the optimization of Relational Database Usage involving large datasets, and in turn, applied these strategies successfully to Big Data. In our discovery phase, we determined that the

majority of Big Data analysis currently occurring in private industry is largely dealing with structured datasets sized in the multi-terabyte range [33]. In many cases, the volume of data was hard to handle not because the database engine is incapable of handling it, but because either the data model or the application queries were ill-designed: the developers thought only of how the application would be putting the data INTO the database, and not how the data would be coming back OUT [14].

From these optimization strategies we further generalized a Model Architecture for Big Data Applications using Relational Databases. Effective Big Data applications must handle the retrieval of decisioned results based on stored large datasets efficiently. Currently, the model works comfortably with terabyte-sized data sets, providing in-database analytics on existing database solutions, lowering the volume of data being handled by the application itself, and significantly decreases response wait times while allowing the flexibility and permanence of a standard relational SQL database.

Future work on the Big Data optimization strategies could include developing an optimized distributed partition modeling scheme. This scheme could include increased attribute-splitting in the case of dynamic determination of the in-database analytics cost approaching an unacceptable pre-determined timing threshold.

REFERENCES

[1] Gantz, J., Reinsel, D. "Extracting Value from Chaos," The 2011 IDC Digital Universe study sponsored by EMC, June 2011.

[2] S. Dowdy, S. Wearden. Statistics for Research. pp 230, 1983.

[3] E. Agirre, A. Soroa. "Personalizing PageRank for Word Sense Disambiguation," EACL '09 Proceedings of the 12th Conference of the European Chapter of the Association for Computational Linguistics, pp. 33-41, 2009.

[4] E. Agirrey, E. Alfonsecaz, K. Hallz, J. Kravalovazx, M. Pascaz, A. Soroay. "A Study on Similarity and Relatedness Using Distributional and WordNet-based Approaches," NAACL '09 Proceedings of Human Language Technologies: The 2009 Annual Conference of the North American Chapter of the Association for Computational Linguistics, pp.19-27, 2009.

[5] D. Bollegala, Y. Matsuo, and M. Ishizuka, "A Web Search Engine-Based Approach to Measure Semantic Similarity between Words," IEEE Transactions On Knowledge And Data Engineering, Vol. 23, No. 7, pp. 977-990, July 2011.

[6] Abu-halaweh, N.M. & Harrison, R.W. "FDT 1.0: An improved fuzzy decision tree induction tool," Fuzzy Information Processing Society (NAFIPS), 2010 Annual Meeting of the North American, pp. 1-5, 2010.

[7] Frank, A. & Asuncion, A. (2010). UCI Machine Learning Repository [http://archive.ics.uci.edu/ml]. Irvine, CA: University of California, School of Information and Computer Science.

[8] Sanghi, S. "Determining the Membership Values to Optimize Retrieval in a Fuzzy Relational Database," Association of Computing Machinery (ACMSE 2006), Proceedings of the 44th annual Southeast regional conference, pp. 537-542, 2006.

[9] Data Over Cable Service Interface Specifications. Cable Labs. Retrieved Jan 02, 2014, from http://www.cablelabs.com/specifications/CM-SP-PHYv3.0-I08-090121.pdf

[10] Sommerville, I. (2011) Software Engineering, 9th Edition. Boston, Massachusetts: Pearson Education, Inc.

[11] Davis, A. "Fifteen Principles of Software Engineering." IEEE Software 11.6 (1994): 94+.

[12] ANSI/ISO/IEC International Standard (IS). Database Language SQL—Part 2: Foundation (SQL/Foundation). 1999.

[13] United States Department of Health and Human Services (2013, Oct 20). Doing Better: Making Improvements to HealthCare.gov. [Web log post] . Retrieved Mar 02, 2014, from http://www.hhs.gov/digitalstrategy/blog/2013/10/making-healthcare-gov-better.html.

[14] Jacobs, A. "The Pathologies of Big Data," ACM Queue: Tomorrow's Computing Today. vol. 7, issue 6, pp. 1-10, July 2009.

[15] "Guide to SQL Programming: SQL:1999 and Oracle Rdb V7.1." Oracle. Retrieved Mar 02, 2014, from http://www.oracle.com/technetwork/database/database-technologies/rdb/0307-sql1999-130211.pdf

[16] SQL99, SQL/MM, and SQLJ: An Overview of the SQL Standards. Databases and Information Systems - Institute of Computer Science - University of Innsbruck.

Retrieved Mar 02, 2014, from http://dbis-informatik.uibk.ac.at/files/ext/lehre/ss11/vo-ndbm/lit/ORel-SQL1999-IBM-Nelson-Mattos.pdf

[17] Eisenberg, A. (1996). "New standard for stored procedures in SQL". ACM SIGMOD Record 25 (4): 81–88.

[18] ISO/IEC 9075:1992 - Information technology -- Database languages -- SQL. ISO - International Organization for Standardization. Retrieved March 2, 2014, from http://www.iso.org/iso/iso_catalogue/catalogue_ics/catalogue_detail_ics.htm?csnumber=16663

[19] MySQL 5.7 FAQ: Stored Procedures and Functions. MySQL Reference Manual. Retrieved March 2, 2014, from http://dev.mysql.com/doc/refman/5.5/en/faqs-stored-procs.html

[20] CALL statement. IBM iSeries Information Center, Version 5 Release 3. Retrieved March 2, 2014, from http://publib.boulder.ibm.com/infocenter/iseries/v5r3/index.jsp?topic=/db2/rbafzmstcallstmt.htm

[21] Stored Procedures and Functions. Oracle Documentation. Retrieved March 1, 2014, from http://docs.oracle.com/cd/B28359_01/server.111/b28318/data_access.htm#CNCPT1776

[22] PostgreSQL: Documentation: 9.3: Procedural Languages. PostgreSQL: The world's most advanced open source database. Retrieved March 1, 2014, from http://www.postgresql.org/docs/current/interactive/xplang.html

[23] "Stored procedure." Sysbase Wiki. Retrieved March 1, 2014, from http://www.petersap.nl/SybaseWiki/index.php/Stored_procedure

[24] "Stored Procedures (Database Engine)." Microsoft Technet: SQL Server. Retrieved March 1, 2014, from http://technet.microsoft.com/en-us/library/ms190782.aspx

[25] Genotype-Phenotype Datasets. Stanford University HIV Drug Resistance Database. Retrieved March 12, 2014, from http://hivdb.stanford.edu/cgi-bin/GenoPhenoDS.cgi

[26] Object Relational Impedance Mismatch. Wikipedia. Retrieved March 1, 2014, from http://en.wikipedia.org/wiki/Object-relational_impedance_mismatch

[27] Zadeh, L. "Common Sense Reasoning Based on Fuzzy Logic," Proceedings of the 18th conference on Winter simulation (ACM), pp. 445-447, 1986.

[28] "Lotfi A. Zadeh" faculty page from College of Engineering, Electrical Engineering and Computer Science, University of California at Berkeley. UC Berkeley EECS. Retrieved March 1, 2014, from http://www.eecs.berkeley.edu/Faculty/Homepages/zadeh.html

[29] From search engines to question answering systems—The problems of world knowledge, relevance, deduction and precisiation, Fuzzy Logic and the Semantic Web, Elie Sanchez (Ed.), Elsevier, Ch. 9, 163-210, 2006.

[30] Zadeh, L. "Quantitative fuzzy semantics." Information Sciences: an International Journal 3.2 (1971): 159-176.

[31] PostgreSQL: Documentation: 9.3: Procedural Languages. PostgreSQL: The world's most advanced open source database. Retrieved March 1, 2014, from http://www.postgresql.org/docs/current/interactive/xplang.html

[32] Pavlo, Andrew, Erik Paulson, Alexander Rasin, Daniel J. Abadi, David J. DeWitt, Samuel Madden, and Michael Stonebraker. "A Comparison of Approaches to Large-scale Data Analysis." Proceedings of the 2009 ACM SIGMOD International Conference on Management of Data (SIGMOD '09) (2009): 165-78.

[33] McKendrick, Joseph. Big Data, Big Challenges, Big Opportunities: 2012 IOUG Big Data Strategies Survey. Rep. Oracle, Inc, Sept. 2012. Web. 01 July 2014.

APPENDICES

APPENDIX A

Appendix A.1 List of Papers Published as a Graduate Student

- "Optimization of Relational Database Usage Involving Big Data"

  to be presented at the 2014 IEEE Symposium on Computational Intelligence and Data Mining (CIDM 2014), 12/09/2014 - 12/12/2014 in Orlando

  Sponsored by the IEEE Computational Intelligence Society, Data Mining Committee

  http://cis.ieee.org/data-mining-tc.html

- "FDT 2.0: Improving scalability of the fuzzy decision tree induction tool - integrating database storage"

  to be presented at the 2014 IEEE Symposium on Computational Intelligence in healthcare and e-health, Workshop on Big Data Analytic Technology for Bioinformatics and Health Informatics (CICARE 2014), 12/09/2014 - 12/12/2014 in Orlando, Florida

  Sponsored by the IEEE Computational Intelligence Society, eCare and eHealth

  http://www.cs.stir.ac.uk/events/CICARE2014/bigdata.htm

- "A Model Architecture for Big Data applications using Relational Databases"

  presented at the IEEE International Conference on Big Data 2014, Workshop on Complexity for Big Data (C4BD2014), 10/27/2014 - 10/30/2014 in Washington, DC

  Sponsored by the IEEE International Conference on Big Data

  http://cci.drexel.edu/bigdata/bigdata2014/index.htm

- "A Novel Approach to Determine Docking Locations Using Fuzzy Logic and Shape Determination" (Poster)

presented at the IEEE International Conference on Big Data 2014, IEEE Big Data 2014 POSTER (IEEEBigData2014), 10/27/2014 - 10/30/2014 in Washington, DC

Sponsored by the IEEE International Conference on Big Data

http://cci.drexel.edu/bigdata/bigdata2014/index.htm

- "A Novel Approach to Determine Docking Locations Using Fuzzy Logic and Shape Determination"

  published with the proceedings of the IEEE International Conference on Big Data 2014, IEEE Big Data 2014 Poster Session Papers (IEEEBigData2014), 10/27/2014 - 10/30/2014 in Washington, DC

  Sponsored by the IEEE International Conference on Big Data

  http://cci.drexel.edu/bigdata/bigdata2014/index.htm

- "A Novel Approach to Determine Docking Locations using Fuzzy Shape Recognition"

  to be presented at 2015 ICS Conference (2015 INFORMS): DMOS: Data Mining and Open Source Software tracks, 1/11/2015 - 1/15/2015 in Richmond, VA

  Sponsored by the INFORMS Computing Society (ICS)

  http://go.vcu.edu/ics2015