A publication of ISCA*:
International Society for Computers
and Their Applications

INTERNATIONAL JOURNAL OF COMPUTERS AND THEIR APPLICATIONS

TABLE OF CONTENTS

*** "International Journal of Computers and Their Applications is abstracted and indexed in INSPEC and Scopus."**

Volume 24, No. 1, March 2017 **ISSN 1076-5204**

International Journal of Computers and Their Applications

A publication of the International Society for Computers and Their Applications

ISCA Headquarters···•···64 White Oak Court, Winona, MN 55987···•···Phone: (507) 458-4517
E-mail: isca@ipass.net • URL: http://www.isca-hq.org

Editor's Note: March 2017

It is my distinct honor, pleasure and privilege to serve as the Editor-in-Chief of the International Journal of Computers and Their Applications (IJCA). I have a special passion for the International Society for Computers and their Applications.

I would like to begin this volume by giving a review of this past year. In 2016 we had 30 articles submitted to the International Journal of Computers and Their Applications. We currently have 11 that are still under review. As a reminder, the journal will not be accepting articles that are less than 6 pages. The authors of these papers will be encouraged to submit their papers to ISCA conferences.

We are still working towards getting IJCA online. Hopefully we can end up with a nice repository soon.

I look forward to working with everyone in the coming years to maintain and further improve the quality of the journal. I would like to invite you to submit your quality work to the journal for consideration of publication. I also welcome proposals for special issues of the journal. If you have any suggestions to improve the journal, please feel free to contact me.

Frederick C. Harris, Jr.
Computer Science and Engineering
University of Nevada, Reno
Reno, NV 89557, USA
Phone: 775-784-6571
Email: Fred.Harris@cse.unr.edu

This year we have 4 issues planned (March, June, September, and December). We begin with a special issue from the best papers at the ISCA Fall Conference cluster (CAINE, and SEDE). We have a proposal for the best papers from the ISCA Spring Conference cluster (CATA/BICOB) which will appear in the September or December issue. The other two issues are being filled with submitted papers.

I would also like to announce that I begun a search for a few Associate Editors to add to our team. There are a few areas that we would like to strengthen our board with, such as Image Processing. If you would like to be considered, please contact me via email with a cover letter and a copy of your CV.

Frederick C Harris, Jr.
Editor-in-Chief
Email: Fred.Harris@cse.unr.edu

Guest Editorial:
Special Issue from ISCA Fall--2016 SEDE Conference

This Special Issue of IJCA is a collection of five refereed papers selected from the SEDE 2016: 26th International Conference on Software Engineering on Data Engineering, held during September 26-28, 2016, in Denver, Colorado, USA.

Each paper submitted to the conference was reviewed by at least two members of the International Program Committee, as well as by additional reviewers, judging the originality, technical contribution, significance and quality of presentation. After the conferences, five best papers were recommended by the Program Committee members to be considered for publication in this Special Issue of IJCA. The authors were invited to submit a revised version of their papers. After extensive revisions and a second round of review, the five papers were accepted for publication in this issue of the journal.

The papers in this special issue cover a broad range of research interests in the community of computers and their applications. The topics and main contributions of the papers are briefly summarized below.

JUAN C. QUIROZ of Sunway University and SERGIU M. DASCALU of University of Nevada Reno, USA presented a web application for the procedural generation of perturbations of 3D models in their paper "Interactive Shape Perturbation". The perturbations are encoded using GP, with an IGA allowing the user to quickly explore perturbations based on his/her preference. Their Procedural Content Generation (PCG) was implemented as a web application, allowing users to create perturbations on their web browser, without having to install libraries or plug-ins.

WENDY OSBORN and FATEMA RAHMAN of University of Lethbridge, Lethbridge, Alberta, CANADA, proposed and evaluated in their paper "Approximate k-Nearest Neighbour Search with the Area Code Tree" a strategy for approximate k-nearest neighbour searching using the Area Code tree. They found that when the Area Code tree is used for locating approximate nearest neighbours, that low constant-time search is achieved. In addition, in denser POI sets, higher accuracy is achieved for locating one nearest neighbour. This ultimately makes the Area Code tree a strong candidate for approximate continuous nearest neighbor processing for location-based services.

CHASE D. CARTHEN, VINH LE, RICHARD KELLEY, TOMASZ J. KOZUBOWSKI and FREDERICK C. HARRIS JR. of University of Nevada Reno, USA, introduced in their paper "Rewind: An Automatic Music Transcription Web Application" an Automatic Music Transcription (AMT) system named *Rewind* that boasts a new deep learning method for generating transcriptions at the frame level and web application. *Rewind*'s new deep learning method utilizes an encoder-decoder network where the decoder consists of a gated recurrent unit (GRU) or two GRUs in parallel and a linear layer and it allows users to transcribe, listen to, and see their music.

FRANCIS ONODUEZE and SHARAD SHARMA of Bowie State University, Maryland, USA described in their paper "Rijndael Algorithm for Database Encryption on a Course Management System" an implementation of the Rijndael algorithm for database encryption to increase the security of a Course Management System. The benefits and drawbacks of various database encryptions were studied based on the amount of data encrypted and modes of access to keep a balance between efficiency and security. The proposed algorithm was applied on a web interface that accepts users' login details, secures through a thorough encryption process, and stores cipher text in the database.

SALEH M. ALNAELI, MELISSA M. SARNOWSKI, CALVIN MEIER and MARK HALL of University of Wisconsin Colleges, USA presented in their paper "Evolution of the Multicore Adaptability of Scientific Software Systems" an empirical study on the challenges of scientific software system in utilizing the full advantage of modern multicore technologies. Twelve open source scientific systems were studied, comprising over 5.4 million lines of code and containing more than 84.5 thousand for-loop statements. They found that the greatest inhibitor to parallelizing scientific software systems is the presence of function calls with side effects, followed closely by data dependency and jumping statements. The study proposes some software engineering techniques to improve the parallelizability of scientific systems.

As guest editor's we would like to express our deepest appreciation to the authors and the program committee members of the conference these papers were selected from.

We hope you will enjoy this special issue of the IJCA and we look forward to seeing you at a future ISCA conference. More information about ISCA society can be found at http://www.isca-hq.org.

Guest Editors:

Frederick C. Harris, Jr, University of Nevada, Reno, USA, SEDE 2016 Conference Chair
Sergiu M. Dascalu, University of Nevada, Reno, USA, SEDE 2016 Program Chair
Yan Shi, University of Wisconsin-Platteville, USA, SEDE 2016 Program Chair

February 2017

Interactive Shape Perturbation

Juan C. Quiroz*
Sunway University, Bandar Sunway, MALAYSIA

Sergiu M. Dascalu†
University of Nevada, Reno, Reno, NV, USA

Abstract

We present a web application for the procedural generation of perturbations of 3D models. We generate the perturbations by generating vertex shaders that change the positions of vertices that make up the 3D model. The vertex shaders are created with an interactive genetic algorithm, which displays to the user the visual effect caused by each vertex shader, allows the user to select the visual effect the user likes best, and produces a new generation of vertex shaders using the user feedback as the fitness measure of the genetic algorithm. We use genetic programming to represent each vertex shader as a computer program. This paper presents details of requirements specification, software architecture, high and low-level design, and prototype user interface. We discuss the project's current status and development challenges.

Key Words: Vertex shader, interactive genetic algorithm, genetic programming, procedural content generation.

1 Introduction

Procedural content generation (PCG) is the algorithmic creation of game content. PCG provides the potential to reduce the cost and time to create content, while also augmenting the creativity of designers, artists, and programmers [25]. We present a PCG web application that enables users to create perturbations of 3D models. Rather than creating 3D models from scratch or from a set of polygonal primitives, we start with a well-formed 3D model and explore variations of the 3D model. The perturbations are generated with vertex shaders.

A vertex shader allows mathematical operations to be performed on the individual vertices that make up a 3D model [2]. The vertex shader performs operations on each vertex, and thus provides great flexibility to modify the position, color, texture, and lighting of individual vertices. The challenge is that making a vertex shader requires programming and computer graphics experience. In addition, even if a programmer writes a vertex shader that produces interesting results, creating a variation of that vertex shader is not straightforward.

We use an interactive genetic algorithm (IGA) to allow users to explore perturbations of 3D models. In an IGA, a user guides a search process by visually evaluating solutions and providing feedback based on personal preferences [28]. Our web application displays perturbed 3D models, the user selects the perturbation he/she likes the best, and the user feedback is used to generate new perturbations. This process is repeated until the user is satisfied. The IGA generates the vertex shaders using genetic programming (GP) [14]. In GP, computer programs are typically represented as tree structures [5, 14].

This paper makes two contributions. First, we present a web PCG application for exploring perturbations of 3D models. Our web application runs on a web browser without having to install plug-ins or any additional software. Second, we use GP to evolve vertex shaders which perturb the vertices that make up the 3D models. In this paper, we use the terms perturbations and transformations interchangeably.

The remainder of this paper is structured as follows. Section 2 describes background on genetic algorithms and related work. Section 3 lists the functional and nonfunctional software requirements for the system. Section 4 presents the use case diagram and the use cases of the system. Section 5 describes the system's architecture. Section 6 reports on the system's results. Section 7 presents conclusions and future work.

2 Background

Our web application uses an existing 3D model as the seed to explore variations of the 3D model with an IGA. The user thus explores and evaluates vertex shaders by seeing the rendered result of each vertex shader applied to the 3D model and guiding the IGA with subjective feedback. Our implementation relies on genetic algorithms (GAs), interactive genetic algorithms (IGAs), and genetic programming (GP) to create the vertex shaders.

2.1 Genetic Algorithms

A GA is a search algorithm based on the principles of genetics and natural selection [5]. The GA maintains a population of individuals, where each individual is a potential solution to the problem being solved. In our case, each individual consists of a vertex shader program. During

*Computing and Information Systems. Email: juanq@sunway.edu.my.
†Computer Science and Engineering. Email: dascalus@cse.unr.edu.

initialization of the GA, the individuals are created randomly. To generate a new population, parents are selected for crossover, with parent selection favoring the fittest individuals. The resulting offspring are mutated. This process repeats until a terminating condition is met.

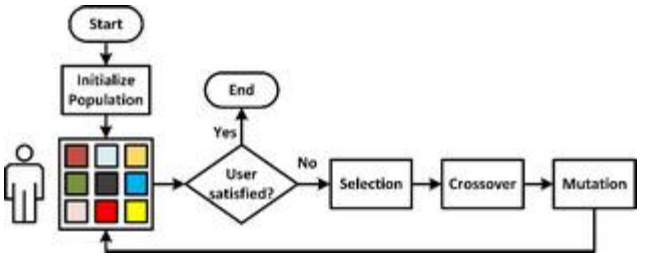


Figure 1: Flowchart of interactive genetic algorithm.

In a GA, fitness evaluation assigns a fitness value to each individual in the population. This is what drives the search process towards higher fitness solutions. When dealing with subjective criteria, such as aesthetics and user preferences, it can be difficult to create an algorithm to determine the fitness of an individual [28]. An IGA solves this by replacing the fitness evaluation with user evaluation, as illustrated in Figure 1. Thus, the IGA presents the user with individuals from the population, the user evaluates the solutions—scores, ranking, selecting the best—and the IGA proceeds with selection, crossover, and mutation.

In the canonical GA, individuals in the population are encoded using a binary string representation. In GP, computer programs are encoded and evolved by the GA [14]. The computer programs are typically represented as tree structures. In our web application, we use GP to create mathematical equations to change the positions of vertices in the vertex shader.

2.2 Texture Evolution

Table 1 provides a summary of current literature in texture, shader, and 3D model evolution. The table indicates whether the evolutionary algorithm was driven by a user, the representation used to encode solutions, the algorithm used, and whether a target image was used to drive the evolutionary process.

Using GAs and GP for evolving images and textures in graphics was done in the early 1990s by Karl Sims [26, 27]. In the work by Sims, an IGA was used for the interactive exploration of images, textures, volume textures, 3D parametric surfaces, and animations. Sims used GP with Lisp symbolic expressions (s-expressions) as the representation. The advantage of Sims' work was that users could create a large variety of complex content without having to understand the underlying equations.

Another technique for automatically generating textures—without user interaction—relies on evolutionary computation to generate textures with characteristics similar to target

Table 1: Summary of texture, shader, and 3D model evolutionary systems in the current literature

Studies	User Interaction	Content Generated	Representation	Algorithm	Target Image
[6]	✓	Textures, volume textures, animations	S-expr	GP	×
[7]	✓	Textures, 3D shapes, 2D dynamical systems	S-expr	GP	×
[8], [9]	✓	Textures	Directed acyclic graphs	GP	✓
[10]	✓	Textures	Directed acyclic graphs	Steady-state GA	✓
[11], [12]	×	Textures	S-expr	GP	✓
[13]	×	Textures	Tree	GP	✓
[14]	×	Textures	S-expr	GP	✓
[15]	×	3D textures	S-expr	GP	✓
[16]	×	Textures	Tree	GP	×
[17]	✓	Vertex and pixel shaders	Byte array	Linear GP	×
[18]	×	Pixel shaders	Tree	GP	×
[19]	✓	Fragment shaders	Opcode numbers, numbers, and bit string	Linear GP	×
[20], [21]	✓	3D shapes	Bit string	GA	×
[22]	✓	Textures	Bit string	GA	✓
[23]	✓	3D Trees	Bit string	GA	×
[24], [25]	✓	Vertex shaders	Bit encoded binary trees	GA	×

images [8-10, 23-24, 30]. The Genshade system evolved high-level Renderman shaders to generate textures similar to a target image [8, 9]. Directed acyclic graphs were used to represent the shaders, with nodes being Renderman shader primitives. In [10], Ibrahim improved the Genshade system by incorporating a knowledge base of intermediary solutions to improve the GA search for textures that shared characteristics with target images. Ibrahim also incorporated Monte Carlo tree search to build new Renderman shaders in parallel to the GA.

In [30], the Gentropy system used GP to generated 2D textures similar to one or more target images. Gentropy used low-level texture generation with mathematical operators, noise, and turbulence effects. In [24], the evolutionary process of Gentropy was improved by including Pareto multi-objective optimization of three features: color, shape, and smoothness. In [23], textures were procedurally generated with GP and Pareto optimization of two objectives. The first objective was a mathematical model of aesthetics derived from fine art. The second objective compared the color distribution of generated images to the target image.

Hewgill and Ross used GP to evolve 3D procedural textures, where the inputs of the procedural function were the X, Y, and Z values of a vertex, and the output was an RGB value for that vertex [6]. A user selected surface points on a model and information was extracted from these surface points: coordinate, surface normal, interpolated mesh normal, and surface gradient. The GA then evaluated each procedural texture by how well its data matched the data from the surface points.

2.3 Vertex Shader Evolution

We refer to evolving vertex shaders as the process of using an evolutionary algorithm, such as GAs or GP, to generate vertex shaders. The outputs of this process are shaders written in a shading language. In [4], Ebner et al. used linear GP to evolve vertex and pixel shaders. Their work was limited to applying texture, color, and lighting values to pixels and vertices, without changing or displacing the positions of vertices. In [7], Howlett et al. used GP to create vertex shaders that applied coloring schemes to scenes of a 3D city environment. In [15], Meyer-Spradow et al. used GP to evolve fragment shaders that applied materials to 3D objects by using bidirectional reflection distribution functions.

In our prior work, we used an IGA to evolve equations in vertex shaders to create perturbations of 3D models [20, 21]. Our prior IGA implementation used Python-Ogre, the python binding for the OGRE graphics rendering engine. The limitations of our prior implementation were that our system (1) was a desktop application requiring users to install a large set of libraries to run the program, (2) used a bit-encoded, full binary tree representation for the individuals in the IGA. In particular, the bit-encoded, full binary tree representation allowed us to encode the equations in an array and to use a standard GA instead of using GP. However, this limited the search space of equations we could generate. The implementation described in this paper uses GP for the evolution of the vertex shaders, allowing the evolutionary process to explore a bigger space of equations.

2.4 Shader Editors

Due to the difficulty of shader programming, content creation tools, such as Blender and Maya, provide graphical shader development tools [11]. In these systems, shaders are created by connecting nodes and building a shade tree. Jensen et al. developed a shader editor that improved traditional shader development by automatically handling transformations between different mathematical spaces and optimizing code placement and space conversions [11].

WebGL has enabled the creation of web based shader editors. ShaderToy and GLSL Sandbox are examples of online pixel shaders editors [1, 12, 19]. In ShaderToy, a scene consist of a fullscreen quad, with the pixel shader written using ray marching and ray casting to create graphics on the quad [12, 19]. Similarly, GLSL Sandbox is a live pixel shader editor. Both ShaderToy and GLSL Sandbox provide a platform for users to publish pixel shaders online, to allow users to edit and bootstrap off public pixel shaders. A disadvantage of ShaderToy and GLSL Sandbox is that there is no geometry in different positions over space and time in the scene. As a result, creating scenes requires implicit programming and a strong use of mathematics to [12, 19].

ShaderFrog is a WebGL shader editor for vertex and fragment shaders applied to 3D objects in a 3D scene [22]. Shaders are created by writing code, by tweaking parameters of public shaders, or by using a graphical user interface to compose a shader that combines the effects of two or more shaders. ShaderFrog also provides a public repository for browsing scenes and shaders created by users.

2.5 Interactive Evolution of 3D Models

IGAs have also been used to evolve the parameters of algorithms that create 3D models. In [17-18], 3D model shapes were created with the implicit surface method, which blended primitive shapes into a complex shape. The GA encoded parameters of operations applied to each primitive, such as scaling, position, rotation, tapering, shearing, twisting, etc. The GA also encoded blending parameters for combining the primitive shapes. In [3], an IGA evolved the parameters used by a procedural tree generation algorithm to create 3D trees.

In this paper, we present the design and implementation of a web application for creating and exploring perturbations of 3D models. Since our system runs on a web browser, there is no need to install additional software or plugins on the client computer. We apply GP to evolve equations that are used to generate vertex shader code that changes the

positions of vertices of a 3D model. In contrast, prior GP has been used to perform operations on vertex data other than position, such as colors, textures, and materials.

3 Requirements Specification

3.1 Functional Requirements

The functional requirements describe the most important behavior of the software, including user interactions and rendering processes.

1. The system shall render a 3D model.
2. The system shall divide the screen into a 3x3 grid, with a viewport for each cell in the grid.
3. The system shall use a camera for each viewport.
4. The system shall display a 3D model within each viewport.
5. The system shall allow the user to move all of the cameras simultaneously with a single set of keyboard controls and the mouse.
6. The system shall allow the user to select with the mouse the 3D model the user likes the best.
7. The system shall allow the user to step the IGA a number of generations.
8. The system shall allow the user to save a selected perturbation of a 3D model.
9. The system shall allow the user to start the IGA.
10. The system shall allow the user to load a 3D model from the file system.
11. The system shall allow the user to browse perturbations stored in a public database.
12. The system shall allow the user to use a perturbation loaded from a public database to seed the IGA.

3.2 Non-Functional Requirements

The non-functional requirements outline the most important constraints on the system.

1. The system shall render in real-time.
2. The system shall be implemented with HTML, JavaScript, and Three.js
3. The system shall run on a web browser with HTML5 support and hardware acceleration.
4. The server request handling shall be implemented with Python.
5. The IGA shall be implemented with the Distributed Evolutionary Algorithms in Python library.
6. The system shall be implemented using a representational state transfer (REST) architecture.
7. The system shall use GP to generate the vertex shader equations.
8. The system shall use 3D models in JSON format.
9. The system shall use time as an input to the vertex shader.
10. The system shall use addition, subtraction, multiplication, division, negation, ceiling, floor, square root, log, sine, and cosine functions in the equations.

4 Use Case Modeling

To gain further insight into the functionality of the system, we have divided the behavior of the web application into use cases. The use case diagram in Figure 2 outlines the controls that allow the user to interact with the system and the functionality on the server-side for running the IGA. In order to further clarify the functionality, detailed descriptions of each use case are presented.

UC01 *Start Evolution:* The user selects the parameters of the IGA. The user then pushes the start button to display the first set of perturbations.

UC02 *Select Model:* The user selects the perturbation the user likes best. The user can select multiple perturbations.

UC03 *Step Generation:* The user submits the selected perturbation by clicking the Next button or by using a keyboard shortcut. This sends a request to the server to assign fitness values to the individuals in the population and create a new population.

UC04 *Camera Control:* The user can move the camera to zoom in to the 3D model or to view the 3D model from a different angle. The camera controls will move all of the viewport cameras at the same time. The keyboard and the mouse can be used to control the camera.

UC05 *Load Model:* The user can upload a 3D model in JSON format using a file browser dialog. After the upload completes, the 3D model will be displayed within each viewport. The user can also browse 3D models uploaded by other users, select one of the models, and load the model to the scene.

UC06 *Save Transformation:* The user can save a transformation by first making a selection and then clicking the save button.

UC07 *Load Transformation:* The user can load a transformation from a file or from a database of perturbations created collaboratively by users over time. Loading the transformation/perturbation will also inject the corresponding vertex shader equation to the IGA population.

UC08 *Render Scene:* The GPU on the client device will render the scene, applying a vertex shader to each of the 3D models.

UC09 *Initialize Population:* The server initializes the population of the IGA in response to receiving a request to start the evolutionary process.

UC10 *Fitness Evaluation:* The selections from the user are received and used to calculate the fitness of the individuals in the IGA population. The fitness of each individual is calculated by determining the similarity between the individual and the user selections.

UC11 *New Population*: The IGA performs selection, crossover, and mutation to generate a new population of

individuals.

UC12 *Select Subset:* Out of the large population size of the IGA, the server selects a subset of vertex shaders to return to the client for rendering on the user's screen.

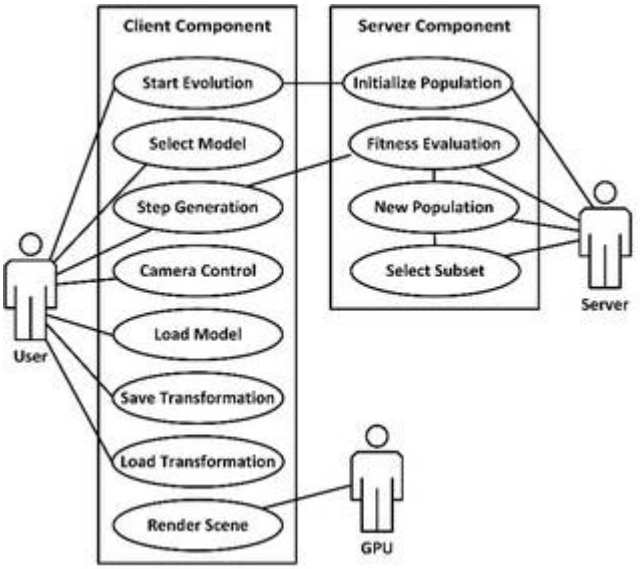


Figure 2: Use case diagram for web application

5 Architectural and Detailed Design

5.1 Architectural Design

Figure 3 illustrates the architecture of the web application. The front-end uses WebGL for rendering graphics on the browser. WebGL is a JavaScript API for rendering 2D and 3D graphics within compatible browsers. Most importantly, no plug-ins need to be installed for the rendering to work. We use Three.js, a 3D JavaScript library, which simplifies the process of writing the WebGL code.

We used the REST architecture for the communication protocol between client and server. The RESTful web services were implemented with Flask, a web microframework for Python. Our RESTful API provides the interface for initializing the IGA population, receiving the user selections, generating a new population, saving perturbations, and loading perturbations and models. The IGA was implemented using the Distributed Evolutionary Algorithms in Python (DEAP) library. The database stores the vertex shaders saved by users and uploaded models.

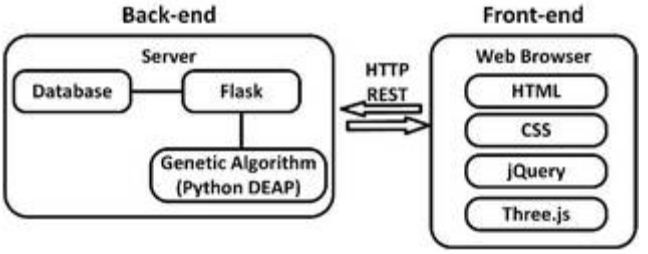


Figure 3: RESTful architecture for web application

5.2 System Activity Chart

Figure 4 presents an activity chart of the web application, showing the interaction between the user and the IGA. The user starts the evolution, which sends a GET request to the server to retrieve vertex shaders to apply to the models currently rendered on the user's web browser. The IGA maintains a large population size, and from this large population, a subset is selected to be evaluated by the user [21]. The user then has the option of selecting a model, and submitting this as fitness feedback to the IGA, with fitness evaluation done as in [21]. The user can repeat this process for as many generations as he/she wants.

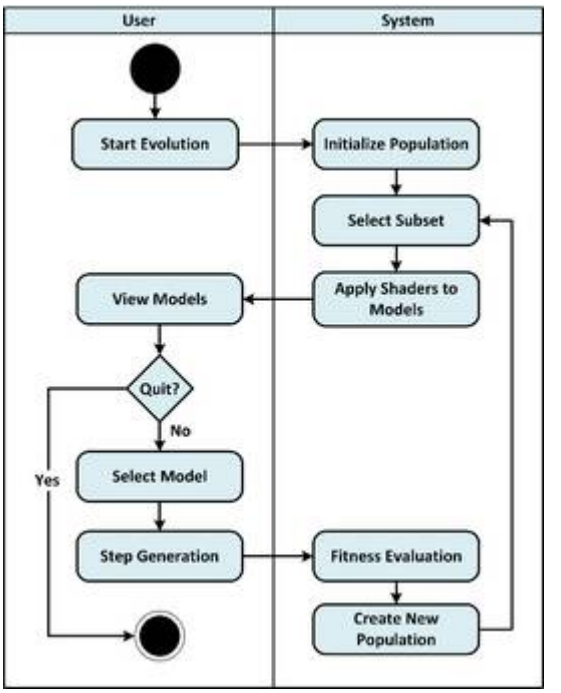


Figure 4: Activity chart for user interaction IGA

5.3 Low Level Design

Figure 5 illustrates an example tree structure generated using GP. The tree structure represents the expression: $x / (x + z)$, where x and z are the x-coordinate and the z-coordinate of the current vertex—we used *x* and *z* instead of *position.x* and *position.z* for brevity and readability. The internal nodes include the operators of addition, subtraction, multiplication, division, negation, ceiling, floor, square root, log, and the trigonometric functions of sine and cosine [26]. The leaf nodes include constants uniformly distributed over the half-open interval [-1, 1), a time variable, and the x, y, or z coordinate of the current vertex. The program, such as the one in Figure 5, is evaluated and it results on a scalar value, a delta. This scalar value is added to the x, y, and z coordinates of the current vertex. That is, each x, y, and z are changed by the same delta per vertex. For example, the equation from Figure 5 would be used in the vertex shader

as shown in Equation (1), where "*position.xyz*" represents a vector containing the x, y, and z coordinates of the current vertex.

$$position.xyz += position.x / (position.x + position.z) \quad (1)$$

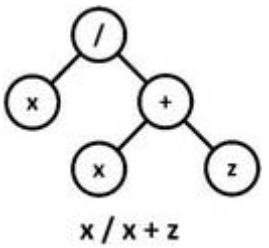


Figure 5: Sample tree structure and its evaluation

Our GP uses a large population size (100), from which we select the best nine perturbations to be evaluated by the user [21]. When the user selects a perturbation as the best, the IGA interpolates the fitness of every other individual in the population based on similarity to the user selected best [21]. The fitness function calculates the fitness of an individual by taking the least square sum between the user selected best (an equation) and the individual (another equation). Since the equations have variables (x, y, z, time), we test 10,000 evenly spaced points in the interval [-10, 10] for the values of these variables. We leave the exploration of alternative fitness functions and tree similarity metrics for future work.

6 Results

Figure 6 shows the interface of our web application. The system loads a default 3D model, consisting of a human character. Users can also upload their own 3D model, which inserts the new 3D model into the scene. A button is provided next to each model to select a model. The scene is divided into viewports, forming a 3x3 grid. Figure 6 shows an example of the interface after 8 generations. Some of the perturbations are quite destructive, such as the perturbations in the middle column of Figure 6. In some cases, the equations do not generate noticeable effects, such as the model in row two – column one.

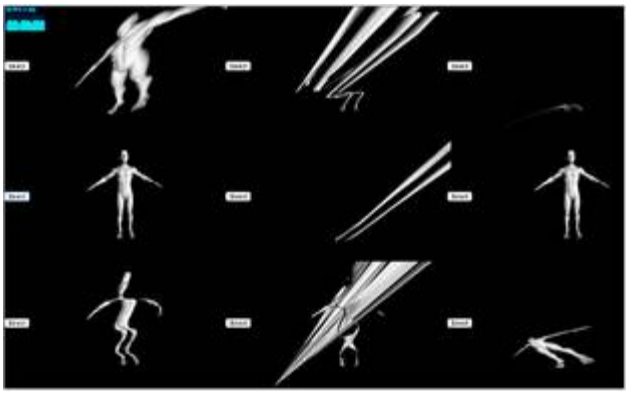
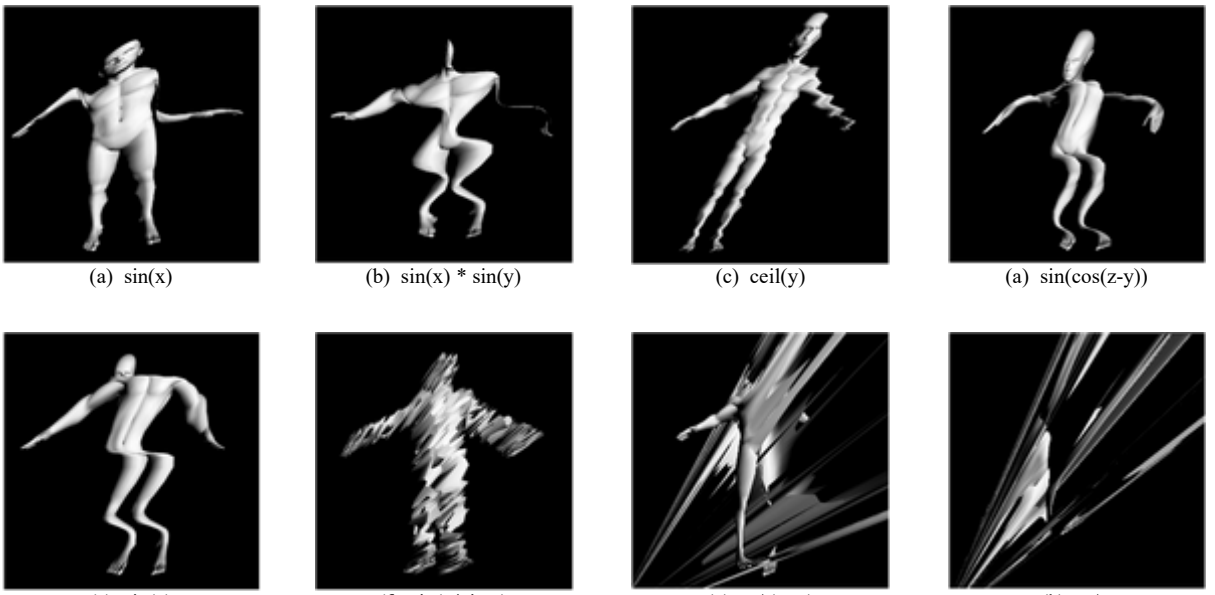


Figure 6: Main interface after evolution has started

Figure 7 illustrates various perturbations generated with our web application. Figures 7(a)-(f) illustrate aesthetically pleasing perturbations, where the shape of the 3D model can still be appreciated. The use of trigonometric functions tends to result in curvy models, giving the models a more finished appearance. The perturbations in Figures 7(g)-(h)



(a) sin(x) (b) sin(x) * sin(y) (c) ceil(y) (a) sin(cos(z-y))

(e) sin(y) (f) sin(y*time) (g) x / (x+z) (h) x / z

Figure 7: Example transformations generated by the corresponding equation in the vertex shader

are quite destructive since the shape of the original model is lost and portions of the model are missing (due to vertices being translated far from the camera view space).

Equations without the time variable result in static perturbations. The disadvantage of static perturbations is that they tend to have noisy and unfinished appearances. The time variable has the potential to make perturbations interesting because of how the model geometry changes each frame. For example, the equation *sin(y * 10.0)* results in a model similar to Figure 7(f). However, Figure 7(f) is a single frame of an animation generated by the equation *sin(y * time)*. Replacing the scalar value *10.0* with the *time* variable makes the noise oscillate, which results in an interesting perturbation.

The sine and cosine functions as the root node of a GP tree result in wavy perturbations. Figure 7(e) shows the perturbation resulting from the equation *sin(y)*. Adding the time variable to this equation, *sin(y + time)*, makes the model undulate over time. Adding a multiplier to the time variable—*sin(y + 3.0 * time)*—makes the model undulate faster or slower depending on the value of the multiplier. However, making a modification such as *sin(y * time)* results in the animated noisy model from Figure 7(f).

The trigonometric equations of cosine and sine in the root node of the GP tree results in the modification to each vertex being in the range [-1, 1]. Hence, the change to each vertex is small, while still allowing for interesting results and variety. In contrast, perturbations that do not use the trigonometric functions have the potential to be destructive by exploding the model (moving vertices to positions far from the camera), see Figure 7(g)-(h). We note that a GP tree having the sine or cosine functions as child nodes in the GP tree can also result in destructive perturbations.

7 Conclusions and Future Work

We presented a PCG system for evolving perturbations of 3D models. The perturbations are encoded using GP, with an IGA allowing the user to quickly explore perturbations based on his/her preference. Our PCG system was implemented as a web application, allowing users to create perturbations on their web browser, without having to install libraries or plug-ins. In contrast to the online shader editors such as ShaderToy, GLSL Sandbox, and ShaderFrog, our system enables users to create vertex shaders without programming experience. All of the programming nuances are hidden from users. In fact, users do not even need to know what a vertex shader is.

A limitation of our implementation is the user interaction. IGAs tend to suffer from user fatigue, user boredom, and user inconsistency in the input provided to the GA [28]. Further work is needed to make the user interaction engaging and intuitive to users from generation to generation. A second limitation is the type of equations that can be generated with our GP implementation based on our selection of operators and terminals. The format of equation (1) also introduces a strong limitation on how the GP equations can modify the vertex data. We plan to test changing each of the x, y, and z coordinates of each vertex with different expressions. Finally, our fitness evaluation needs to be improved by using tree similarity instead of sum of least squares.

References

[1] Ricardo Cabello, "GLSL Sandbox Gallery," [Online], Available: http://glslsandbox.com. [Accessed: 18-Jan-2017].

[2] "The Cg Tutorial - Chapter 1. Introduction." [Online]. Available: http://http.developer.nvidia.com/CgTutorial/cg_tutorial_chapter01.html, [Accessed: 15-Jun-2016].

[3] C. B. Congdon and R. H. Mazza, "GenTree: An Interactive Genetic Algorithms System for Designing 3D Polygonal Tree Models," *Proceedings of the 2003 International Conference on Genetic and Evolutionary Computation: PartII*, Berlin, Heidelberg, pp. 2034-2045, 2003.

[4] M. Ebner, M. Reinhardt, and J. Albert, *Evolution of Vertex and Pixel Shaders*, *Genetic Programming*, M. Keijzer, A. Tettamanzi, P. Collet, J. van Hemert, and M. Tomassini, Eds., Springer Berlin Heidelberg, pp. 261-270, 2005.

[5] D. E. Goldberg, *Genetic Algorithms in Search, Optimization, and Machine Learning*, Addison-Wesley Professional, 1989.

[6] A. Hewgill and B. J. Ross, "Procedural 3D Texture Synthesis using Genetic Programming," *Comput. Graph.*, 28(4):569-584, Aug. 2004.

[7] A. Howlett, S. Colton, and C. Browne, "Evolving Pixel Shaders for the Prototype Video Game Subversion," *AI and Games Symposium (AISB'10)*, 2010.

[8] A. E. M. Ibrahim, *Genshade: An Evolutionary Approach to Automatic and Interactive Procedural Texture Generation*, Texas A&M University, 1998.

[9] A. E. M. Ibrahim, "Evolutionary Techniques for Procedural Texture Automation," Advances in Visual Computing, pp. 623-632, 2013.

[10] A. E. M. Ibrahim, "Multiple-Process Procedural Texture," *Vis. Comput.*, pp. 1-18, Aug. 2016.

[11] P. D. E. Jensen, N. Francis, B. D. Larsen, and N. J. Christensen, "Interactive Shader Development," *Proceedings of the 2007 ACM SIGGRAPH Symposium on Video Games*, New York, NY, USA, pp. 89-95, 2007.

[12] P. Jeremias and I. Quilez, "Shadertoy: Learn to Create Everything in a Fragment Shader," *SIGGRAPH Asia 2014 Courses*, New York, NY, USA, p. 18:1-18:15, 2014.

[13] T. Kagawa, H. Nishino, and K. Utsumiya, "The Development of Interactive Texture Designing Method for 3D Shapes," 2010 International Conference on P2P, Parallel, Grid, Cloud and Internet Computing, pp. 109-114, 2010.

[14] J. R. Koza, *Genetic Programming: On the Programming of Computers by Means of Natural Selection*, The MIT Press, 1992.

[15] J. Meyer-Spradow and J. Loviscach, "Evolutionary

Design of BRDFs," *Eurographics 2003 Short Paper Proceedings*, pp. 301-306, 2003.
[16] D. P. Muni, N. R. Pal, and J. Das, "Texture Generation for Fashion Design Using Genetic Programming," Robotics and Vision 2006 9th International Conference on Control, Automation, pp. 1-5, 2006.
[17] H. Nishino, H. Takagi, S. B. Cho, and K. Utsumiya, "A 3D Modeling System for Creative Design," 15th International Conference on Information Networking, pp. 479-486, 2001.
[18] H. Nishino, H. Takagi, and K. Utsumiya, "A Digital Prototyping System for Designing Novel 3D Geometries," 6th Int. Conf. on Virtual Systems and MultiMedia (VSMM2000*)*, pp. 473-482, 2000.
[19] I. Quilez and P. Jeremias, "Shadertoy BETA," 2009, [Online], Available: https://www.shadertoy.com/, [Accessed: 21-Dec-2016].
[20] J. C. Quiroz, A. Banerjee, and S. J. Louis, "3-D Modeling Using Collaborative Evolution," *Proceedings of the Fourteenth International Conference on Genetic and Evolutionary Computation Conference Companion*, New York, NY, USA, pp. 653-654, 2012.
[21] . C. Quiroz, A. Banerjee, S. J. Louis, and S. M. Dascalu, *Collaborative Evolution of 3D Models*, *Design Computing and Cognition '14*, J. S. Gero and S. Hanna, Eds., Springer International Publishing, pp. 493-510, 2015.
[22] A. Ray, "ShaderFrog," *ShaderFrog*, [Online], Available: http://shaderfrog.com, [Accessed: 18-Jan-2017].
[23] B. J. Ross, W. Ralph, and H. Zong, "Evolutionary Image Synthesis Using a Model of Aesthetics," 2006 IEEE International Conference on Evolutionary Computation, pp. 1087-1094, 2006.
[24] B. J. Ross and H. Zhu, "Procedural Texture Evolution using Multi-Objective Optimization," *New Gener. Comput.*, 22(3):271-293, Sep. 2004.
[25] N. Shaker, J. Togelius, and M. J. Nelson, *Procedural Content Generation in Games: A Textbook and an Overview of Current Research*, Springer, 2016.
[26] K. Sims, "Artificial Evolution for Computer Graphics," *Proceedings of the 18th Annual Conference on Computer Graphics and Interactive Techniques*, New York, NY, USA, pp. 319-328, 1991.
[27] K. Sims, "Interactive Evolution of Equations for Procedural Models," *Vis. Comput.*, 9(8):466-476, Aug. 1993.
[28] H. Takagi, "Interactive Evolutionary Computation: Fusion of the Capabilities of EC Optimization and Human Evaluation," *Proc. IEEE*, 89(9):1275-1296, 2001.
[29] A. L. Wiens and B. J. Ross, "Gentropy: Evolutionary 2D Texture Generation," *Late* Breaking Papers at the 2000 Genetic and Evolutionary Computation Conference, Las Vegas, Nevada, USA, pp. 418-424, 2000.
[30] A. L. Wiens and B. J. Ross, "Gentropy: Evolving 2D textures," *Comput. Graph.*, 26(1):75-88, Feb. 2002.

Juan C. Quiroz is a Senior Lecturer in the Department of Computing and Information Systems at Sunway University, Bandar Sunway, Malaysia. He received his Ph.D. in Computer Science and Engineering from the University of Nevada, Reno, USA in 2010. His research interests are in evolutionary computation and machine learning.

Sergiu Dascalu is a Professor in the Computer Science and Engineering Department at the University of Nevada, Reno, USA. He received a PhD in Computer Science from Dalhousie University, Halifax, Nova Scotia, Canada in 2001. His main research interests are in software engineering and human-computer interaction, in particular in software approaches and tools for scientific research, simulation environments, virtual reality, information visualization, and interaction design. Sergiu has over 150 peer-reviewed publications and has been involved in the organization of many international conferences and workshops. He has advised 5 PhD and 40 Master students who graduated so far and has worked on many projects funded by federal agencies such as NSF, NASA, and ONR as well as industry organizations.

Approximate k-Nearest Neighbor Search with the Area Code Tree

Wendy Osborn* and Fatema Rahman*
University of Lethbridge, Lethbridge, Alberta T1K 3M4, CANADA

Abstract

In this paper, we propose and evaluate a strategy for approximate *k*-nearest neighbor searching using the Area Code tree. The Area Code tree is a trie-type structure that manages area code representations of each point of interest (POI) in a data set. It provides a fast method for locating an approximate nearest neighbor for a query point. We first summarize the area code generation, insertion (used in overall construction) and searching approaches. Then, we evaluate the construction via repeated insertion, *k*-nearest neighbor searching, and accuracy of the Area Code tree, with some of the evaluation involving the comparison versus a basic benchmark brute force approach. We find that when the Area Code tree is used for locating approximate nearest neighbors, that low constant-time search is achieved. Also, in denser POI sets, higher accuracy is achieved for locating one-nearest neighbor. This ultimately makes the Area Code tree a strong candidate for approximate continuous nearest neighbor processing for location-based services.

Key Words: Nearest neighbor queries, spatial access methods, location-based services.

1 Introduction

A location-based service provides results to a user of a mobile device (e.g. smartphone, tablet) based on their location, interests and the type of query being performed [11]. One example of such a query is a *k*-nearest neighbor query [10, 12], which returns the nearest *k* points of interest (POI) to them. For example, a user may want to know the location of some of the nearest restaurants to them. The user may want to know the exactly closest restaurants to them. However, the user may also be happy with suggestions that, although not guaranteed to be the closest, may be close enough to satisfy them. This is an example of an approximate *k*-nearest neighbor, where a trade-off is being made between accuracy and efficiency.

Efficient nearest neighbor processing, exact or approximate, is important, but is especially important when it is initiated from a mobile device [11]. Many strategies have been proposed for nearest *k*-neighbor processing for location-based services. Several utilize spatial access methods [1], including [3, 4, 5, 6, 7, 13, 14]. Although all of these strategies return exact nearest neighbors, limitations of these approaches include repeated searching, the need to cache a significant amount of data on the mobile device, the requirement to know the query trajectory in advance, and maintaining a sparse index which leads to inefficient searches.

Repeated searching is not desirable, but may be the only option when storage on a mobile device is limited. A recently proposed data structure, the Area Code tree [8] manages POIs in a trie-type structure using an area code representation for each POI. Although it can be used to locate POIs efficiently, it cannot be used for exact nearest neighbor matching. However, given a preliminary evaluation on its efficiency, it is an excellent candidate for approximate *k*-nearest neighbor search for situations where a guaranteed exact answer is not required, for example, in the restaurant scenario given above.

Therefore, in this paper, we extend this approximate nearest neighbor strategy and propose one for locating *k*-nearest neighbors. We also evaluate the Area Code tree for *k*-nearest neighbor search accuracy, tree construction time, and also comparatively evaluate its search time against another strategy. We find that approximate *k*-nearest neighbor searching can be accomplished in very low and constant time, regardless of the number of POIs being indexed. With respect to accuracy, up to 60% accuracy is achieved when the Area Code tree is used for locating one-nearest neighbor in a dense POI set. This makes the Area Code tree a significant candidate for continuous approximate nearest neighbor search for location-based services. The remainder of the paper proceeds as follows. Section 2 summarizes related work in the area of continuous nearest neighbor processing for mobile devices. Section 3 summarizes the area code mapping, insertion and 1-nearest neighbor search algorithm for the Area Code tree. Given this, we propose a *k*-nearest neighbor search strategy for the Area Code tree. Section 4 presents the methodology and results of our performance evaluation. Finally, Section 5 concludes the paper and provides some directions of future research.

2 Related Work

In this section, we summarize related work in nearest neighbor searching for location-based services. Although nearest neighbor strategies have been proposed in other

*Email: {wendy.osborn,f.rahman}@uleth.ca. Article is an extension of [9], presented at SEDE 2016.

contexts, they are considered outside of the scope of this work. Many strategies have been proposed in the literature [3, 4, 5, 6, 7, 13, 14].

A continuous nearest neighbor strategy proposed by Song and Roussopoulos [13] obtains a superset *m* of nearest neighbors, which attempts to keep the result current while the query point moves around, and a new query call to the server is not necessary. Their strategy utilizes existing stationery nearest neighbor strategies. A limitation of their strategy is in choosing the value of *m* so that fewer query calls are needed but not too much data needs to be stored on the mobile device.

Tao *et al.* [14] utilizes the R-tree [2] to speed up the repeated searching needed for their continuous query strategy. Lee *et al.* [5] improves upon this strategy by fetching both the required and some additional objects in order to reduce the number of repeated searches that are needed. Park *et al.* [7] also improves upon this strategy by locating all nearest neighbors along a trajectory by using the R-tree. A limitation exists in that the trajectory needs to be known in advance.

Hu *et al.* [3] proposes a proactive caching strategy, which caches previous results and the R-tree nodes required to obtain them on the mobile device. The cache is always searched first for a new query, with additional results fetched from the server. Limitations include the significant overhead of caching and local processing on the mobile device.

Jung *et al.* [4] utilize a grid index for continuous nearest neighbor searching. The grid index allows for quick elimination of regions of space from consideration if they do not overlap the query point. A limitation with this strategy is that the grid index can be sparse due to wasted space.

In summary, some general limitations of these approaches include caching a significant amount of data on the mobile device, repeated searching, using a sparse index which leads to inefficient searches, and requiring knowledge of the query path in advance. Repeated searching, however, may be the only option if storage is limited on a mobile device. The Area Code tree [8] attempts to provide the ability to perform repeated searching that is efficient, but at the cost of accuracy. It is also more compact that existing spatial access methods, given that only POIs are stored, and not many co-ordinates for many bounding rectangles. The Area Code tree is summarized in the next section.

3 Area Code Tree

The Area Code Tree [8] is a trie-type structure for approximate nearest neighbor searching. It stores points of interest (POIs) that are represented in an area code format. In this section, we summarize the mapping, construction and nearest neighbor search for the Area Code tree.

3.1 Mapping

An area code is a sequence of digits that indicate the relative location of a POI in space. It is obtained by recursively partitioning the space containing points into quadrants. For a particular POI, the space is partitioned until the POI is equal to the middle of a quadrant. At each level of partitioning, the quadrants are numbered as follows: SW (1) SE (2) NW (3) and NE (4). Beginning with the top-most partition, a POI obtains a digit at each level, depending on which quadrant it resides in. Figure 1 depicts an example of space partitioning and mapping. The POI maps to the area code 4132, since the POI resides in the top-most NE quadrant, followed at the next level by the SW quadrant, then the NW quadrant, and finally the SE quadrant.

3.2 Tree Construction

Once the area code for a POI is determined, is it inserted into the Area Code Tree, beginning with the most significant area code digit. To construct a tree, each area code is inserted one at

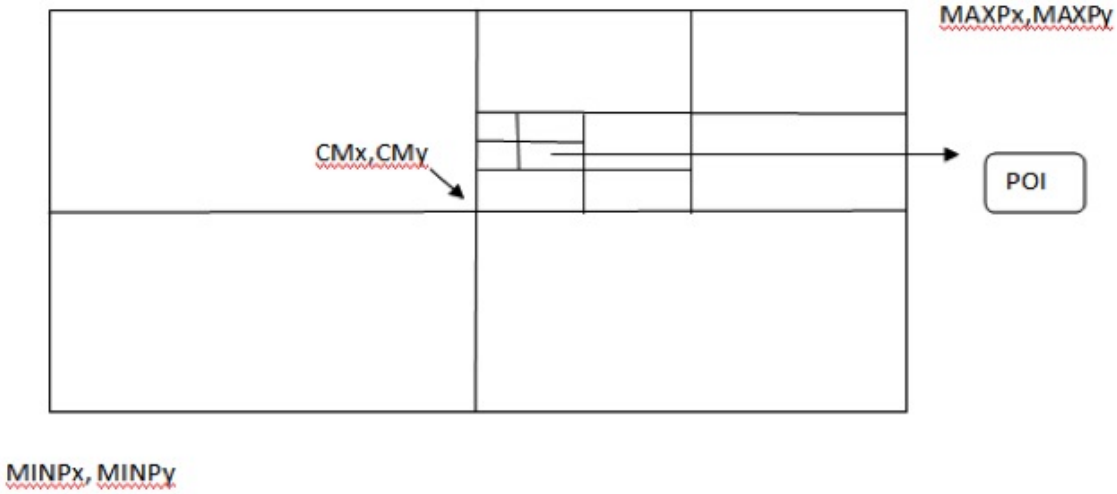


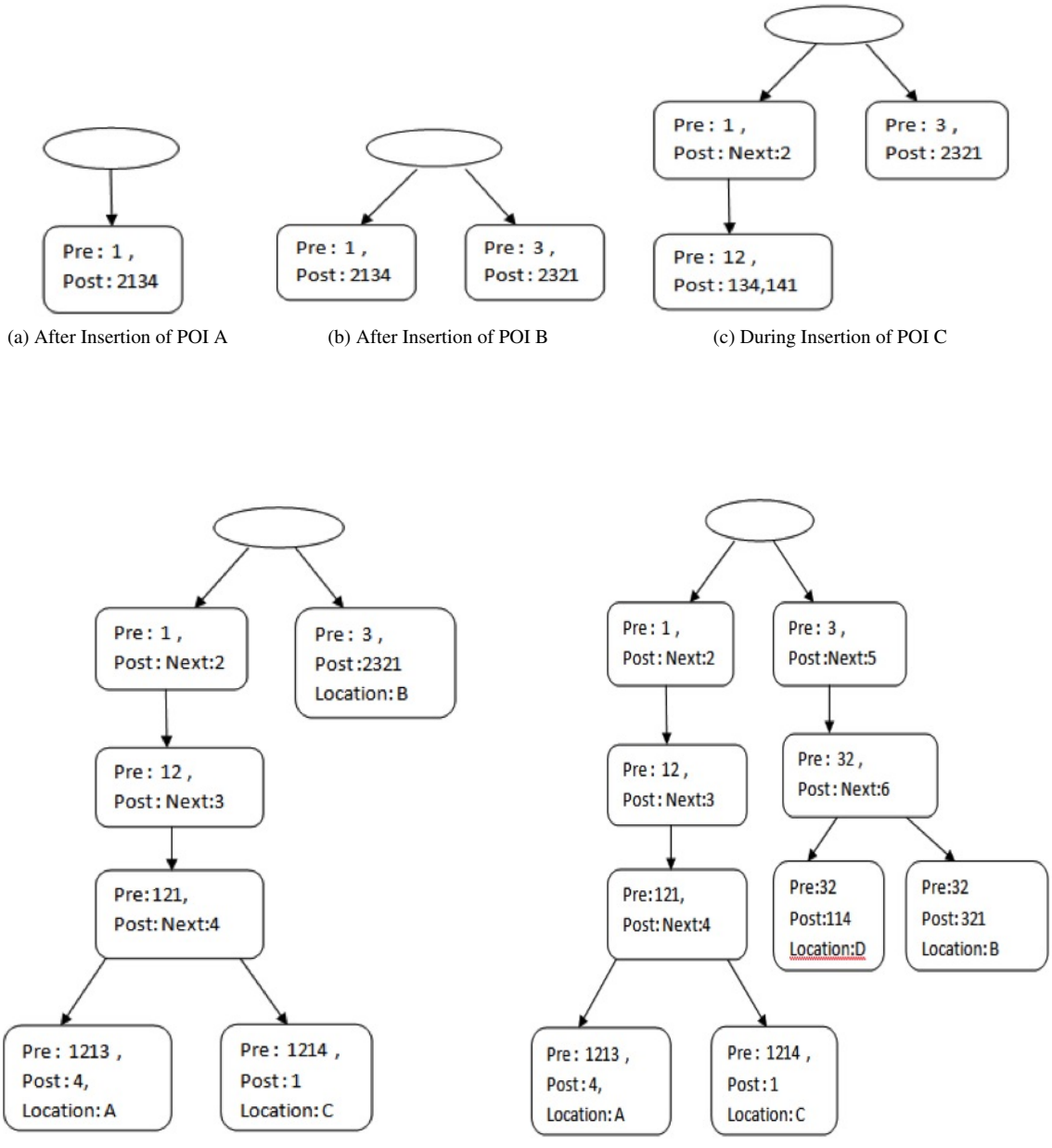
Figure 1: Area code mapping (from [8])

a time, using the existing strategy for any trie structure. Figures 2 and 3 depict the construction of a small Area Code Tree, containing the POIs A,B,C,D and E, and each with area codes 12134, 32321, 12141, 32114, and 21324 respectively. POI A is inserted first. When POI B is inserted, a new node is created for it, since the existing node contains area codes that start with 1, and the area code for B starts with 3. When POI C is inserted, additional nodes are created for the common prefix strings (12, 121) contained by both A and C. A similar case occurs when inserting POI D, with common prefix string 32. Finally, when POI E is inserted, a new path is started since its leading digit, 2, is not contained by any other existing path.

3.3 K-Nearest Neighbor Searching

Given that every POI is mapped to a string of digits, this provides a method for quickly identifying approximate k-nearest neighbors to the query point. We first describe the 1-nearest neighbor case, and then describe the extensions to k nearest neighbors.

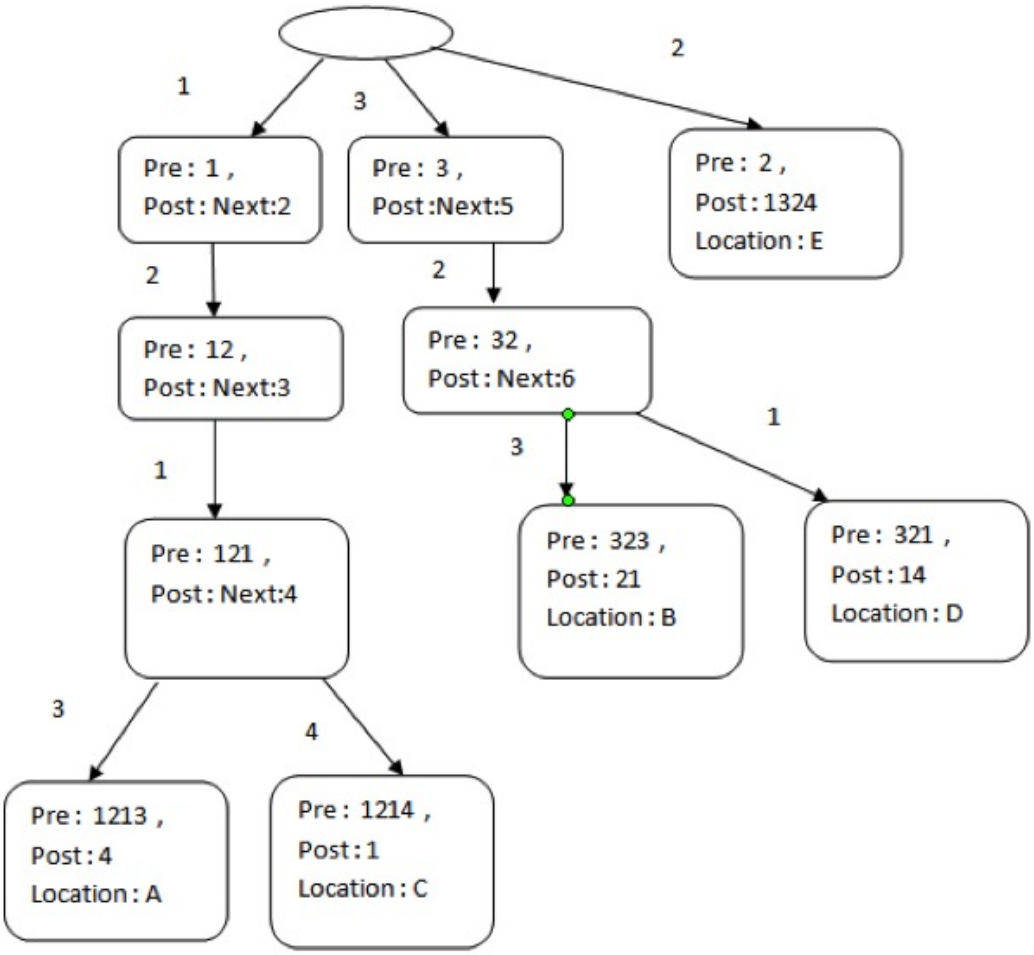
Any search will begin by first mapping the query point to an area code using the same strategy that is mentioned above for



(a) After Insertion of POI A (b) After Insertion of POI B (c) During Insertion of POI C

(d) After Insertion of POI C (e) After Insertion of POI D

Figure 2: Index construction - part 1



(a) After Insertion of POI E

Figure 3: Index construction - part 2

mapping POIs. Then, beginning with the most significant digit of the query point area code, a path is followed down the Area Code tree while digits in the query match digits in the tree nodes. If a match does not exist at a particular level, the closest match is taken. For example, suppose we have the query area code of 12144. Referring back to Figure 3, the closest match is the POI C with area code 12141, since a path exists that matches the first 4 digits of the query area code, with the closest node to the last digit in the query area code containing the value of 1.

Using this basis, the 1-nearest neighbor search can be extended to locate k nearest neighbors. Figure 4 depicts a sketch of our k-nearest neighbor strategy for the Area Code tree. It consists of three stages, which we describe below:

1. Finding nearest neighbors. This stage uses a slight modification of the 1-nearest neighbor approach. A search is initiated for the area code that is the closest to the query. However, if k nearest neighbors are sought, then area codes for other POIs that exist along the immediate search path are also obtained and added to the result set, until k is reached, or until the leaf node for the original search is reached.
2. Finding nearby neighbors. It is possible that the number of POIs obtained in the first step is not enough to satisfy the query. Therefore, using the area code for each POI in the initial result set, a search for other POIs is performed, with any added to the updated result set. This continues until either k is reached, or all POIs in the initial result set have been exhausted.
3. Finding further neighbors. This step is similar to the second step, except that the POIs obtained in the second step are the basis for locating additional nearby POIs.

Referring back to our original example, suppose we have the query area code of 12144. In Figure 3, the closest match is the POI with area code C (12141). However, in the first step of our k-nearest neighbor strategy, the POI A is also retrieved for our result. If k=2, then the search is completed. Otherwise, the result set containing (C,A) is processed using the second step of our strategy to obtain any more required points from the

```
knn = n; //provided by User
query = xxxxx; //converted from (x,y)
              //provided by User
result = list();

tree_ptr = AC_root;

//first, find "matching" area code for query,
//and all others along same path
result
   = find_nearest_neighbors(tree_ptr,
      query,knn);

//if we need more POIs to make inn
if(size(result) < knn)
     result
        = find_nearby_neighbors(tree_ptr,
           query, knn, result)

//if we still need more POIs,
//we look even further from query
if(size(result) < knn)
     result
        = find_further_neighbors(tree_ptr,
           query, knn, result)

return result;
```

Figure 4: *k*-nearest neighbor searching sketch

remaining part of the tree.

4 Evaluation

In this section, we present the methodology and results of our performance evaluation of the Area Code tree. We compare its search performance with the Brute Force method, which consists of searching the entire set of POIs to find the *k*-nearest neighbors. We chose this comparison for our preliminary evaluation due to the Brute Force method being a basic benchmark for processing nearest neighbor queries. In addition, we evaluate the construction time and the accuracy of our proposed structure. Construction is performed by inserting one POI area code at a time, while accuracy is measured by determining the percentage of times overall that accurate *k*-nearest neighbors are found when the Area Code tree is used to locate them.

4.1 Methodology

For our evaluation, we use twenty-one synthetically generated data sets that represent collections of different POIs across New Zealand. Ten data sets consist of POIs drawn from across the North Island of New Zealand, with each set containing 1000, 2000, 3000,..., up to 10,000 POIs respectively. An additional ten data sets contain POIs from across the Waikato Region of New Zealand (part of the North Island), again with each file containing 1000, 2000, 3000,..., up to 10,000 POIs respectively. The Waikato data sets are denser, which allows us to evaluate the Area Code Tree in denser POI data. The remaining data set contains 10 User locations along their trajectory, which will serve as the *k*-nearest neighbor queries for our evaluation.

First, for each POI set mentioned above, an Area Code Tree is created, to give a total of 20 area code trees. Then, using these trees and the User location set, the following tests are performed:

- For all 20 area code trees, we perform a 1-nearest neighbor search for each of the 10 points in the User location set.
- For the North Island 10,000 and Waikato 10,000 POI sets only, we also perform a *k*-nearest neighbor search for each point in the User location set. We perform the search for *k*=2,3,4,...10.

Every search is also performed on the same data sets using the Brute Force method. Therefore, a total of 400 *k*-nearest neighbor comparisons are performed.

The performance criteria that are measured are as follows:

- For each tree construction, the overall construction time (in seconds), where the construction time includes the time required to calculate the area codes and to insert each area code into the tree.
- For every *k*-nearest neighbor search, the average search time (in milliseconds).
- The accuracy of the Area Code tree, which is measured by recording for each search, the percentage of POIs found by Area Code Tree that matched those found by the Brute Force search.

4.2 Results

We first present the results of the search and accuracy experiments, followed by the tree construction results. First, Figures 5 and 6 depict the results of the 1-nearest neighbor comparison using all of the the Waikato and North Island POI sets, respectively. For both figures, the x-axis contains values that represent 1000s of POIs (i.e. 1 is 1000 POIs, up to 10 for 10,000 POIs), while the y-axis contains the average search time in seconds.

We find that for both the Waikato and North Island POI sets, searching using the Area Code Tree achieves significantly better search times over the Brute Force method. The average search time for the Area Code tree is less than 0.005 seconds (i.e. 5ms), and is regardless of both the density of the dataset and the number of POI area codes in the area code tree. For the Brute Force method, the average search time increases linearly,

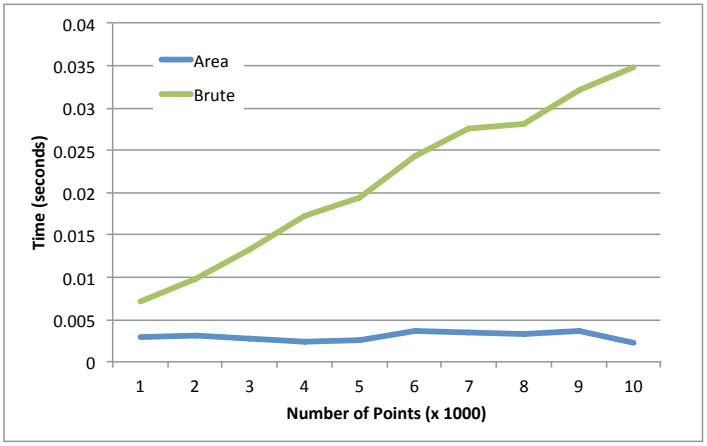


Figure 5: 1-nearest neighbor - Waikato POIs

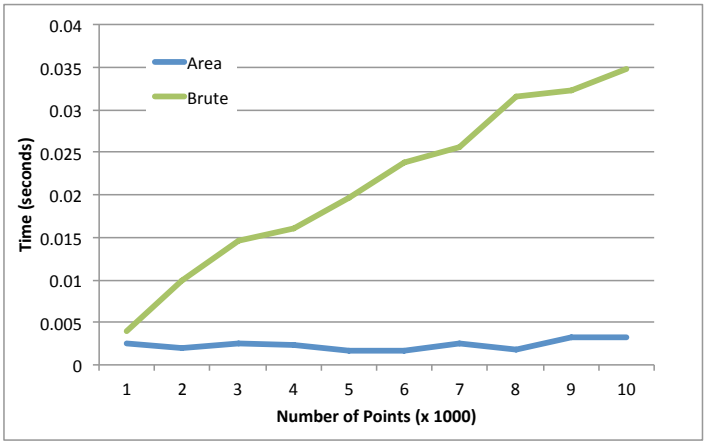


Figure 6: 1-nearest neighbor - North Island POIs

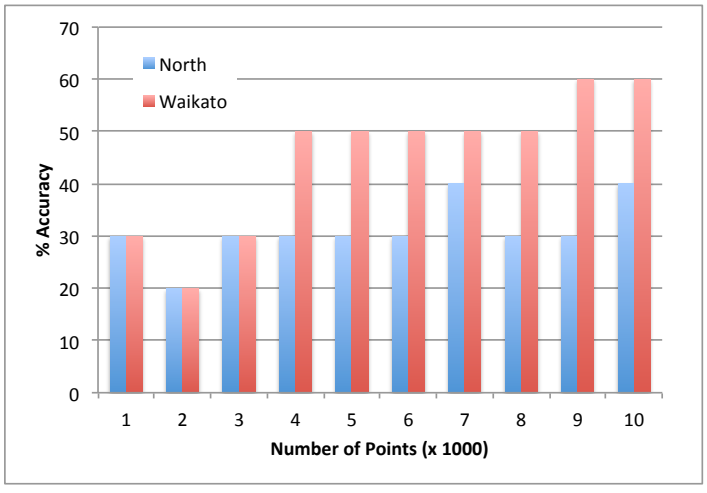


Figure 7: 1-nearest neighbor - accuracy

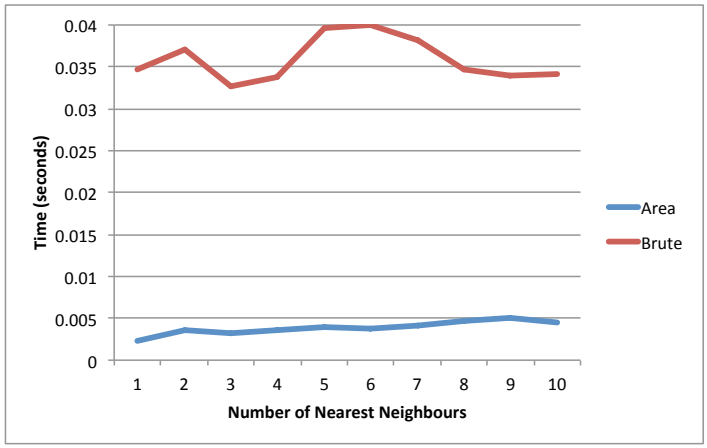


Figure 8: k-nearest neighbor - Waikato POIs

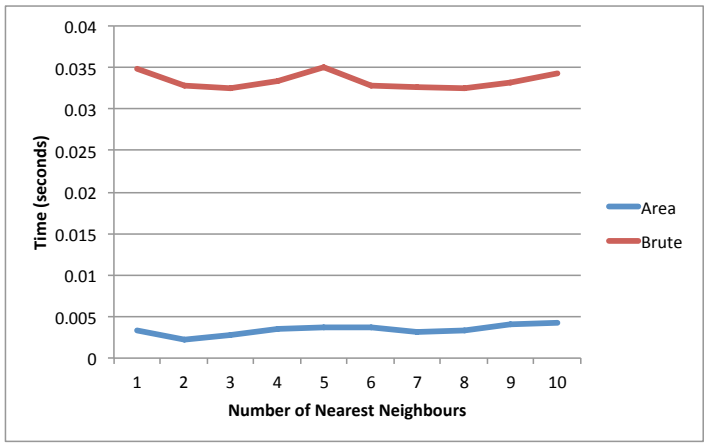


Figure 9: k-nearest neighbor - North Island POIs

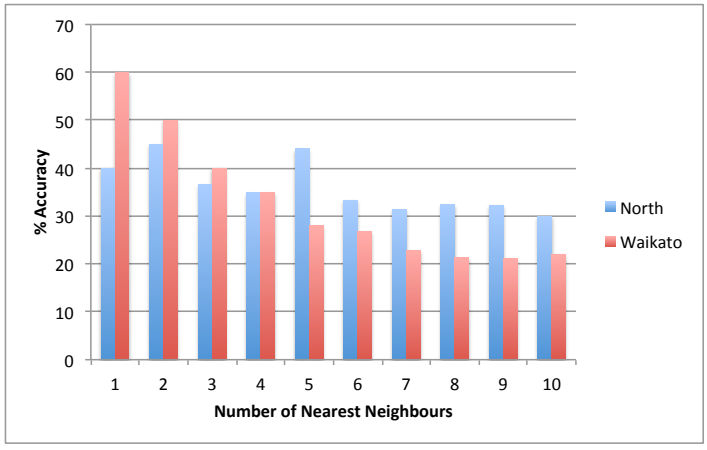


Figure 10: k-nearest neighbor - accuracy

between approximately 0.007 seconds (i.e. 7ms) up to almost 0.035 seconds (i.e. 35ms) for searching in 10,000 area codes.

Figures 8 and 9 depict the results of the *k*-nearest neighbor comparison for the 10,000 Waikato and 10,000 North Island POI sets, respectively. For both figures, that x-axis contains the *k* values used, while the y-axis contains the average search time in seconds. We again find in both cases a significant improvement in running time when the Area Code tree is used for *k*-nearest neighbor searching. We also find that for both Area Code tree searching and the Brute Force approach, similar results are achieved, regardless of the number of nearest neighbors being sought or the density of the POI set. The average search time using the Area Code tree is under 0.005 seconds (i.e. 5ms), while for the Brute Force approach it is between 0.03 and 0.04 seconds (i.e. 30 and 40ms).

Figures 7 and 10 depict the accuracy of the Area Code tree in locating 1-nearest neighbor and *k*-nearest neighbors, respectively. Here, we can see that the North Island and Waikato POI sets differ significantly in their performance.

For 1-nearest neighbor search, we find that for the dense POI sets (i.e Waikato), the Area Code tree can achieve between 40% and 60% accuracy . For the less dense POI sets (i.e. North Island) however, the accuracy was lower, around 30% in most cases. Therefore, for the 1-nearest neighbor case we conclude that the Area Code Tree provides fast nearest neighbor searching, that is expected to improve in accuracy as the data set size increases in density.

However, we find the opposite scenario when the search is increased to *k*-nearest neighbors. For the less dense POI sets, although some modest improvements are found in accuracy, for most cases, and in particular for higher values of *k*, we find that just over 30% accuracy is still being achieved. For the denser POI sets, we observe a steady decline in the accuracy of the search results, from 60% for 1-nearest neighbor to just over 20% accuracy for 10-nearest neighbors. In addition, it should be noted that, although not presented here, the only situations where 100% accuracy is achieved are for the 1- and 2-nearest neighbor searches. Therefore, we conclude that in less dense POI sets, searching in the Area Code tree seems to achieve consistent accuracy, regardless of the *k* value, while for the dense point sets, the best results are for lower values of *k* only, with a decline in accuracy as the value of *k* increases.

Finally, Table 1 depicts the overall construction times (in seconds) of the Area Code Tree for the various sets of POIs. We observe that the time it takes to construct an Area code tree via repeated insertion increases significantly with the size of the POI set. However, the absolute worst case is still under 3 minutes. For the North Island datasets, the time ranges from just a few seconds for 1000 POIs, up to 2.5 minutes for 10,000 POIs. For the Waikato Region datasets, the times range from a few seconds to over 3 minutes. Although these overall construction times are high when compared to the search times, two things must be noted. First, for static data sets, the Area Code tree only needs to be constructed once in order to be searched many times. Second, the occasional insertion can still be performed without having to re-construct the entire Area Code tree. Therefore, given the search performance results above, the construction time is a small price that must be paid.

Table 1: Tree construction times (in seconds)

Data Sets	#POIs	O/A Time
North Island	1000	4.04
	2000	10.98
	3000	19.60
	4000	31.46
	5000	44.31
	6000	61.20
	7000	81.80
	8000	103.61
	9000	127.37
	10000	158.24
Waikato	1000	5.02
	2000	12.63
	3000	23.38
	4000	37.45
	5000	55.19
	6000	75.97
	7000	99.50
	8000	127.75
	9000	159.02
	10000	192.71

5 Conclusion

In this paper, we present a *k*-nearest neighbor search strategy for the Area Code tree. We also evaluate the accuracy, tree construction time, and also comparatively evaluate the search time against a Brute Force strategy. We find that approximate *k*-nearest neighbor searching can be accomplished in very low and constant time, regardless of the number of POIs being indexed. With respect to accuracy, up to 60% accuracy is achieved when the Area Code tree is used for locating 1-nearest neighbor in dense POI sets. This makes the Area Code tree a significant candidate for continuous approximate nearest neighbor search for location-based services. The only significant factor is in the tree construction time. However, for fairly static data sets, this is a small price to pay for the savings in search time and increased accuracy (in some cases) that are achieved. Some future research directions include the following. First, the Area Code tree must have its *k*-nearest neighbor search performance evaluated further by comparing it with other spatial access methods. Second, further examination of the Area Code tree for continuous *k*-nearest neighbor searching must also take place. Finally, the Area Code tree needs to be expanded to utilize other types of "area codes", such as telephone area codes, and postal/zip codes. However, the results presented here show that the Area Code tree has promise for different applications in continuous query processing.

Acknowledgments

The authors would like to thank the reviewers of this manuscript and the previous conference paper for their thoughtful and constructive comments.

References

[1] V. Gaede and O. Günther, "Multidimensional Access Methods," *ACM Computing Surveys*, 30:170–231, 1998.

[2] A. Guttman, "R-trees: A Dynamic Index Structure for Spatial Searching," *Proc. ACM SIGMOD Int'l Conf. Management of Data*, pp. 47–57, 1984.

[3] H. Hu, J. Xu, W.S. Wong, B. Zheng, D.L. Lee and W.-C. Lee, "Proactive Caching for Spatial Queries in Mobile Environments," *Proc. 21st Int'l Conf. Data Engineering*, pp. 403–414, 2005.

[4] H.R. Jung, S.-W. Kang, M.B. Song, S.J. Im, J. Kim and C.-S. Hwang, "Towards Real-Time Processing of Monitoring Continuous k-Nearest Neighbor Queries," *Proc. 2006 Int'l Conf. Frontiers of High Performance Computing*, pp. 11-20, 2006.

[5] K.C.K. Lee, W.-C. Lee, H.V. Leong, B. Unger and B. Zhang,"Efficient Valid Scope Computation for Location-Dependent Spatial Queries in Mobile and Wireless Environments," *Proc. 3rd Int'l Conf. Ubiquitous Information Management and Communication*, pp. 131-140, 2009.

[6] W. Osborn and A. Hinze, "TIP-tree: A Spatial Index for Traversing Locations in Context-Aware Mobile Access to Digital Libraries," *Pervasive and Mobile Computing*, 15:26–47, December 2014.

[7] Y. Park, K. Bok and J. Yoo, "An Efficient Path Nearest Neighbor Query Processing Scheme for Location-Based Services," *Proc. 17th Int'l Conf. Database Systems for Advanced Applcations*, pp. 123–129, 2012.

[8] F. Rahman and W. Osborn, "The Area Code Tree for Nearest Neighbor Searching," *Proc. 2015 IEEE Pacific Rim Conf. Communications, Computers and Signal Processing*, pp. 153–158, 2015.

[9] F. Rahman and W. Osborn, "The Area Code Tree for Approximate Nearest Neighbor Searching in Dense Point Sets," *ISCA 25th Int'l Conf. Software Engineering and Data Engineering*, pp. 103–108, 2016.

[10] N. Roussopoulos, S. Kelley and F. Vincent, "Nearest Neighbor Queries,", *SIGMOD Rec.*, 24(2):71–79, May 1995.

[11] J.H. Schiller and A. Voisard, Editors, *Location-Based Services*, Morgan Kauffman, 2004.

[12] S. Shekhar and S. Chawla, *Spatial Databases: A Tour*, Prentice Hall, 2003.

[13] Z. Song and N. Roussopoulos, "K-Nearest Neighbor Search for Moving Query Point," *Proc. 7th Int'l Symp. Advances in Spatial and Temporal Databases*, pp. 79–96, 2001.

[14] Y. Tao, D. Papadias, and Q. Shen, "Continuous Nearest Neighbor Search," *Proc. 28th Int'l Conf. Very Large Data Bases*, pp. 287–292, 2002.

Wendy Osborn is an Associate Professsor of Computer Science at the University of Lethbridge. She obtained her B.C.S. (Hons) and M.Sc. degrees at the University of Windsor (Ontairo, Canada) in 1996 and 1998 respectively, and her Ph.D. at the University of Calgary (Alberta, Canada) in 2005. Her research interests include location-based services, spatial access methods, and distributed query processing for standard and non-standard data.

Fatema Rahman obtained her B.Sc. degree in Computer Science and Engineering at Jahangirnagar University (Dhaka, Bangladesh) in 2009. Currently, she is a Master of Science (Computer Science) candidate at the University of Lethbridge, and expects to complete her degree in the Summer of 2017. Her fields of interest include data management, mining and warehousing, pattern recognition and image processing.

Rewind: An Automatic Music Transcription Web Application

Chase D. Carthen*, Vinh Le*, Richard Kelley*, Tomasz J. Kozubowski*, Frederick C. Harris Jr.*
University of Nevada
Reno, Nevada, 89557, USA

Abstract

Simple digital audio formats such as mp3s and various others lack the symbolic information that musicians and other organizations need to retrieve the important details of a given piece. However, there have been recent advances for converting from a digital audio format to a symbolic format a problem called Music Transcription. Rewind is an Automatic Music Transcription (AMT) system that boasts a new deep learning method for generating transcriptions at the frame level and web application. The web app was built as a front end interface to visualize and hear generated transcriptions. Rewind's new deep learning method utilizes an encoder-decoder network where the decoder consists of a gated recurrent unit (GRU) or two GRUs in parallel and a linear layer. The encoder layer is a single layer autoencoder that captures the temporal dependencies of a song and consists of a GRU followed by a linear layer. It was found that Rewind's deep learning method is comparable to other existing deep learning methods using existing AMT datasets and a custom dataset. In other words, Rewind is a web application that utilizes a deep learning method that allows users to transcribe, listen to, and see their music.

Key Words: Deep learning, Automatic Music Transcription, Music Information Retrieval, and Machine Learning.

1 Introduction

Many musicians, bands, and other artists make use of MIDI, a symbolic music instruction set, in popular software to compose music for live performances, portability across other formats, and recording. However, most music is often recorded into raw formats such as Wav, MP3, OGG, and other digital audio formats. These formats do not often contain symbolic information, but may contain some form of metadata that does not typically include symbolic information. Symbolic formats, such as sheet music have been used by bands, choirs, and artists to recreate or perform songs. These symbolic formats are effectively the spoken language of music that can be re-translated back into sound. Communities such as Mirex are actively working many different problems on retrieving information from music so that creating, categorizing, and extracting information is easier. The Symbolic format is not only portable, but can be leveraged for doing different types of analysis such as genre classification, artist classification, mood detection.

Automatic Music Transcription (AMT) is the process of converting an acoustic musical signal into a symbolic format [14]. There are a few music transcription applications having varying degrees of accuracy that have been built mostly for Windows, Linux, Mac and the web browser [19]. Only a few of these applications have the ability to visualize the results of the transcription. A piano roll is an intuitive visualization of music that does not require a user to learn a more complex symbolic available for music such as sheet music. These applications allow a user to get a symbolic format of their music that can be used for many different reasons such as changing a song, portability to other applications, live performances, and for generating sheet music. However, most of these applications do not use state of the art algorithms from advances in Deep Learning that have contributed to the Music Information Retrieval (MIR) field.

There has been recent work in the AMT field with [5, 6, 25] that have produced higher transcription accuracies than previous methods. These advances along with the creation of web audio frameworks such as WebAudio or WebMidi have made it possible to playback many different types of audio formats such as mp3, wav, and MIDI. Web frameworks such as Django and Flask make it possible to create a web application that does automatic music transcription and allows users to visualize the transcription and hear the results. Rewind [8, 9] is a tool and method that will make use of a new Deep Learning method based on previous work, visualize the results of the transcribed file, and allow the user to edit the transcribed results.

The following paper is structured as follows: Section 2 covers background related to the MIR and Deep Learning field.

*chase@nevada.unr.edu, vle@nevada.unr.edu
*rkelley@unr.edu, tkozubow@unr.edu
*Fred.Harris@cse.unr.edu

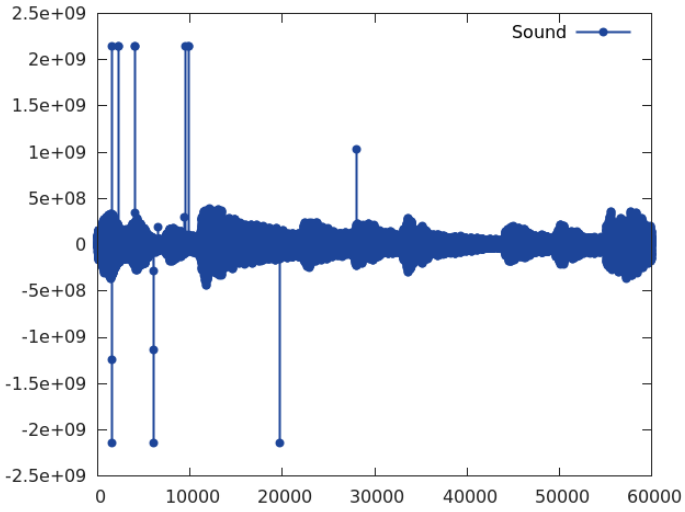


Figure 1: An example of a raw audio file

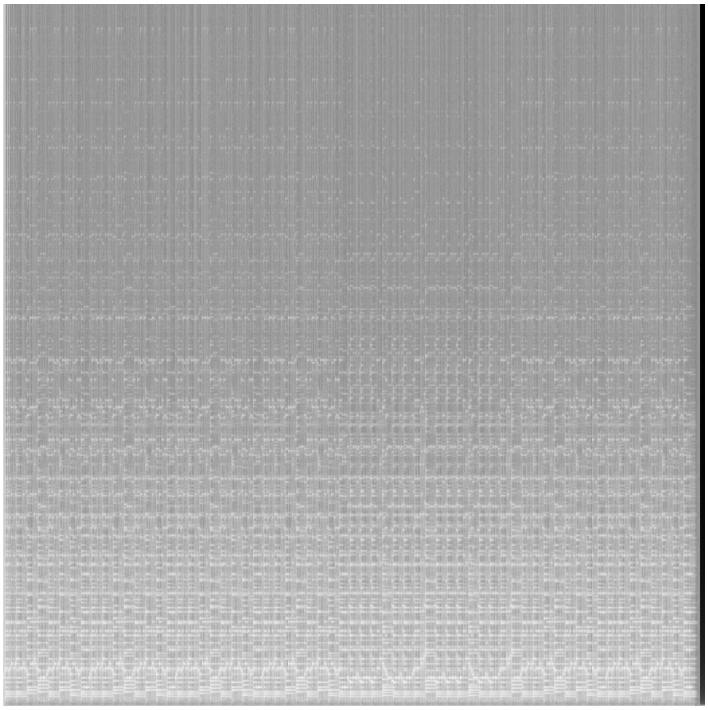


Figure 2: An example of a spectrogram

Section 3 discusses the implementation and design of Rewind tool. Section 4 gives the results of the Rewind method. Finally Section 5 concludes and details future direction that Rewind can take.

2 Background

AMT systems are designed to make transcriptions at the three levels of detail in music, those being the stream, note, and frame level [14]. The stream level is simply a raw acoustic signal which is contained in an audio digital file, an example of which can be seen in Figure 1. The goal of the frame level is to capture all pitches within each frame provided by a spectrogram. An example of a spectrogram is demonstrated in Figure 2. At the note level, a set of pre-existing notes are used to generate a brand new set of notes or create a record of the notes. The note level can be represented as a piano roll or as sheet music. An example of sheet music and piano roll is demonstrated in Figures 3 and 4. Most AMT systems evaluate their effectiveness by means of various metrics, including recall, accuracy, precision, and f-measure [3]. Precision determines how relevant a transcription is given irrelevant entries in a frame. It is defined as follows:

$$Precision = \frac{\sum_{t=1}^{T} TP(t)}{\sum_{t=1}^{T} TP(t) + FP(t)} \quad (1)$$

Recall is the percentage of relevant music transcribed, and is given by Equation 2.

$$Recall = \frac{\sum_{t=1}^{T} TP(t)}{\sum_{t=1}^{T} TP(t) + FN(t)} \quad (2)$$

The accuracy determines the correctness of a transcription, and is given by Equation 3.

$$Accuracy = \frac{\sum_{t=1}^{T} TP(t)}{\sum_{t=1}^{T} TP(t) + FP(t) + FN(t)} \quad (3)$$

While the F-measure determines the overall quality between the precision and recall.

$$F\text{-}measure = \frac{2 * precision * recall}{precision + recall} \quad (4)$$

These metrics in turn are calculated with true positives, false positives, and false negatives

There has been some work using LSTMs and semitone filter banks to transcribe music [5]. In Sigtia's work [25], the idea of an acoustic model converting an audio signal to a transcription is introduced. Additionally this paper introduces using a music language model to improve the accuracy of a transcription of a acoustic model like Boeck [5] and others as well. Boulanger-Lewandowski [6] uses a deep belief network to extract features

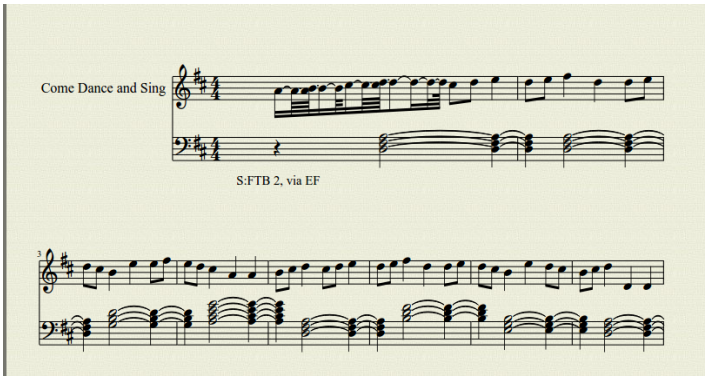


Figure 3: An example of sheet music

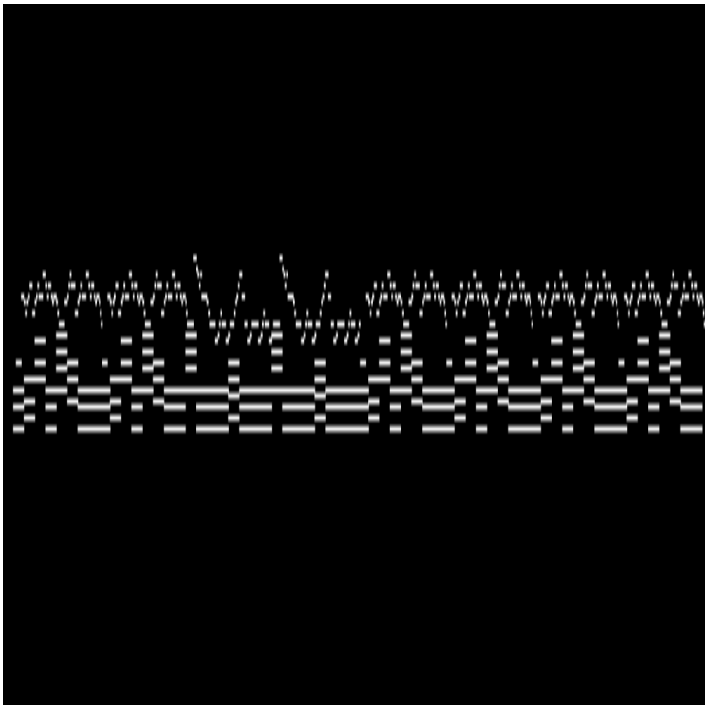


Figure 4: An example of a piano roll

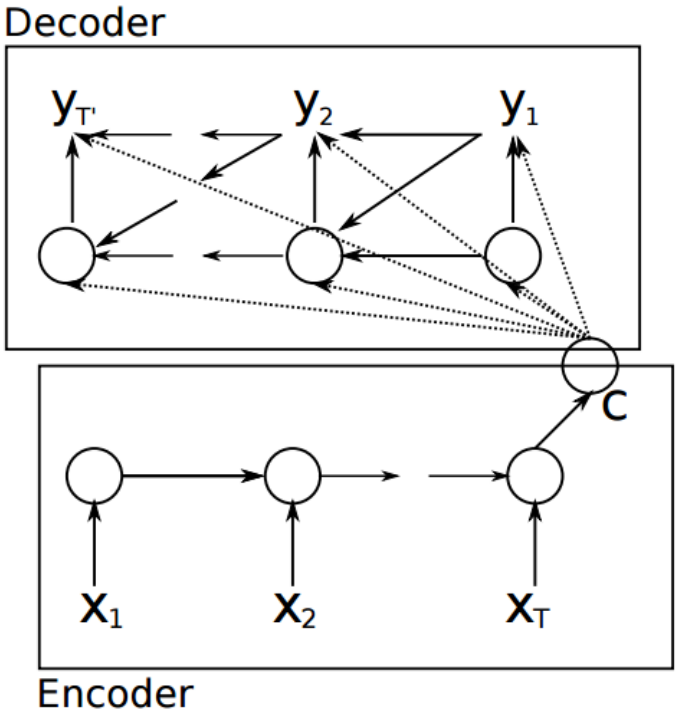


Figure 5: A picture of a encoder-decoder network with a context C demonstrated between the encoder-decoder network [11]

from a spectrogram and utilizes a rnn to create a transcription along with a innovative beam search to transcribe music. Boulanger-Lewandowski's beam search is possible thanks to the generative properties of the deep belief network that is merely a collection of restricted Boltzman machines or RBMs that are stacked. This beam search is also utilized in combination with recurrent neural network with an neural autoregressive distribution estimator (rnn-nade) as a music language model and an acoustic model that uses a deep neural network. A follow-up paper produces a hash beam search that finds a more probable transcription in fewer epochs [24]. Both the beam search and hash beam search produce the most accurate transcriptions.

Recently, encoder-decoder networks have been used for unsupervised learning in terms of autoencoders [26], translation [12], caption generation for images, video clip description, speech recognition [11, 13] or video generation. Autoencoders, like an encoder-decoder network, are commonly used for unsupervised learning to learn features contained inside the data, by using the identity of the data. An autoencoder is powerful for learning features contained within a dataset. However, there are more complex encoder-decoder networks [12, 11, 13], where they learn a context and then map English to French. They are less concerned with learning the identity and more for learning the context of the data presented. Rewind utilizes these types of encoder-decoder networks to learn an encoding for a spectrogram presented to it. An example layout of this network is demonstrated in Figure 5. These networks have proven to be beneficial, and are state of the art.

3 Rewind

Rewind is very much like other AMT systems in that it determines the fundamental frequencies of the notes and what notes are active at the frame level. Like most other frame based systems, Rewind utilizes a spectrogram as its main input and a ground truth midi as the target. All audio samples are constructed at a 22 kHz sample rate and turned into a normalized spectrogram with a 116 ms window size, which can be either a 10 ms or a 50 ms stride. It has been found that a window size larger than 100ms produces the most accurate results with a rnn-lstm [5]. A multitude of existing datasets were utilized for training Rewind's models: Nottingham [1], JSB Chorales [2], Poliner and Ellis [20], Maps [15], MuseData [10], and Piano.midi.de [16]. All of these datasets were split into 70% for training, 20% for testing, and 10% for validation. These datasets consisted of midi only or midi with aligned audio and made into datasets with timidity, Torch's audio library, and a midi library [4]. Rewind's models were implemented with rnn [18] and optim. A simple auto-correlation method was also constructed as a way to implement Rewind's web service and website for quick testing. The auto-correlation is also compared against the encoder-decoder network. Rewind has two types of models: the encoder and the decoder model. The encoder and decoder is very similar to the encoder-decoder network in Figure 5 [11, 12, 13]. The encoder model of Rewind utilizes an autoencoder, which utilizes a single GRU for its encoder, whose

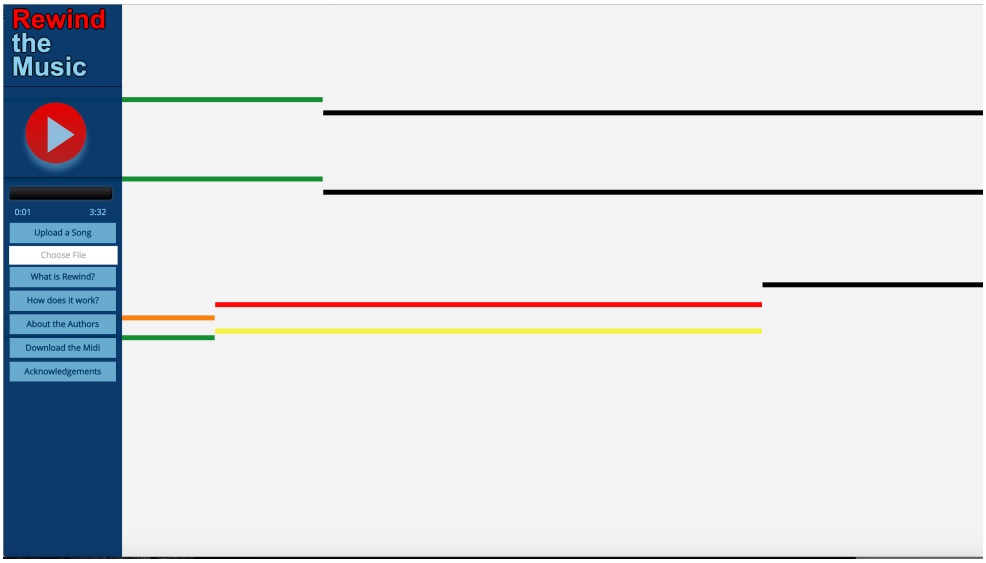


Figure 6: A screenshot of piano roll notes lighting up

output is squashed by a rectified linear unit and a linear layer for its decoding layer. While the decoder model has an identical layout, but its outputs are squashed with a sigmoid activation function and may have a second GRU in parallel.

The encoder network utilizes an autoencoder to create an encoding for a spectrogram. An autoencoder was chosen because a deep neural network (stacked auto encoders) has been used for extracting features from a spectrogram in the case of speech recognition [7] and other similar works that utilize deep belief networks (stacked restricted Boltzman machines) have been used to extract features [17]. A deep belief network, along with an autoencoder, are used to produce a generative model for a spectrogram [13]. The generic representations generated by autoencoders can be further improved with recurrences [26], where the encoder and decoder of the autoencoder are both LSTMs for learning over video sequences and generating video sequences. Rewind's encoder model utilizes a linear neural network for the decoder and a GRU for the encoder with a rectified linear unit (ReLU) for it's activation function [26]. The encoder network is trained with a mean squared error function.

The decoder network consists of two types of networks being a GRU with a linear layer and two GRUs stacked onto each other in parallel with a linear layer. Both types of networks are squashed with a sigmoid function. The GRU in both networks was chosen because it produced the lowest error rate. This network's objective function is binary cross entropy, so that this decoder network will learn a distribution of notes where a probability of one indicates a note on and a probability of zero indicates a note off. Binary cross entropy is used for minimizing the log probability [6, 25], which also utilizes a sigmoid function to create binary probabilities[23]. The binary cross entropy function is demonstrated in Equation 5, where the sum is taken over all distributions [25]:

$$\sum_i t_i \log p_i + (1 - t_i) \log (1 - p_i) \tag{5}$$

The probabilities constructed from the sigmoid function can be used to construct a MIDI, and are utilized in previously mentioned papers. The decoder network's job is to generate these probabilities for each encoding passed by the encoder network.

The auto-correlation method is a very noisy method. The process creates a spectrogram of the required audio file and then each bin of the spectrogram is normalized with the standard deviation and mean. After these transformations have been made, a threshold is applied, where anything greater than the threshold is a 1 and anything less is a 0. Subsequently, one simply only needs to go to each frequency bin that matches a midi note and extract the frequencies that have a value of 1. This auto correlation method is only meant as a test model for a web service. However, in Section 4, results are reported for its accuracy in comparison to Rewind's Network.

3.1 Architecture

Rewind's architecture consists of multiple parts that consist of: the client, models and web service, and the server. Each part is unique and has been designed to handle different parts of Rewind's functionality. The models are used for producing transcription, and the web service is used to interface with the model and send outputs to the client through the server. All visualization, downloads, and uploads are handled by the client. The server pushes all content needed to run the website to the client. An overall diagram of the architecture is demonstrated in Figure 7.

The models and web service component of the architecture are used to process data for training a model, generating

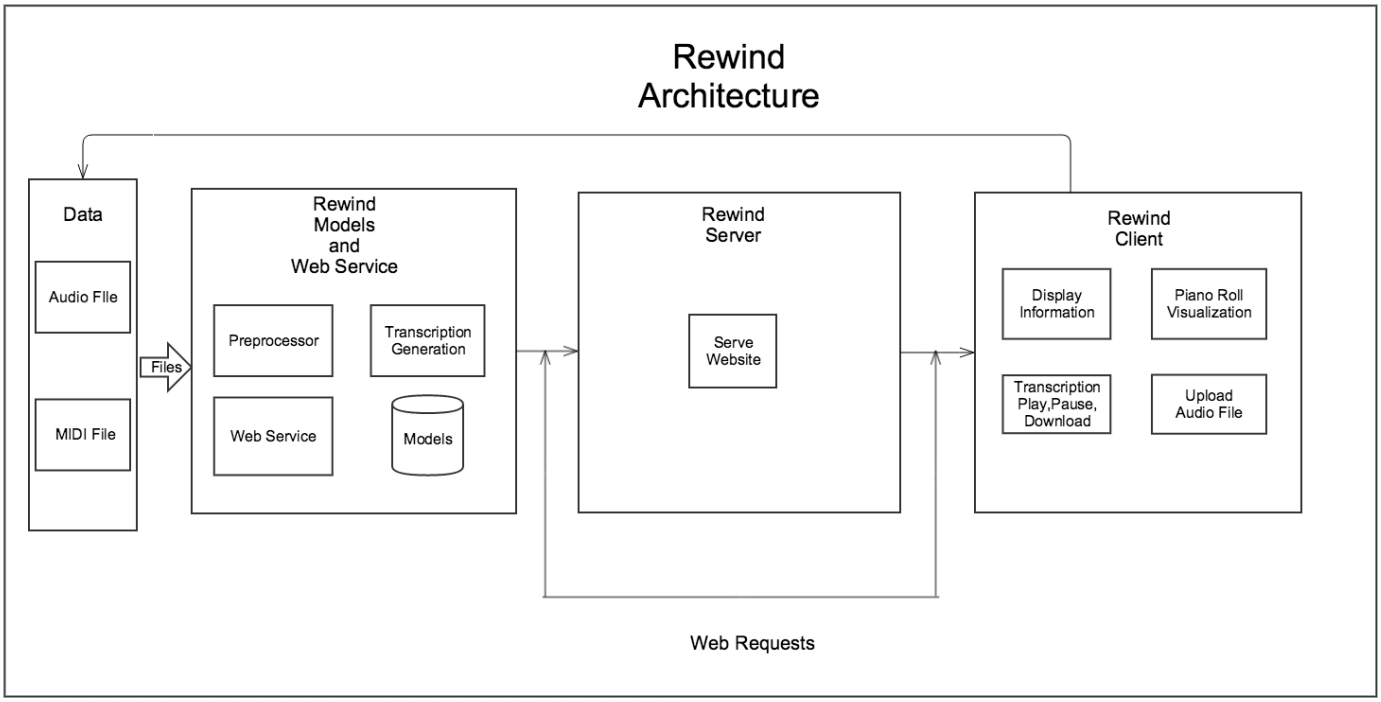


Figure 7: The Architecture of Rewind

transcriptions with a preexisting model to be sent through the web service, and training models. This component contains Rewind's method, or AMT algorithm, for creating transcriptions of digital audio formats. The web service was created as a way for Rewind's models to send transcriptions to the client. The web service for Rewind was written in Flask [22], as it requires a small amount of code to get a web service written.

Rewind's server was created with Django's web framework [21]. The rewind server serves up the website to the client, which includes all of the HTML, Javascript, and CSS files. It also handles sending uploaded audio files to the web service and forwarding the content back to the client.

The client is a web browser, such as Google Chrome or Mozilla Firefox, that is to be utilized by the user. The client handles creating a piano roll for visualization, uploading audio files to the web service, and giving the ability to download a transcription. All sound playback is handled by the client and allows the user to pause and play sounds. The client's job is to light up the notes in the piano roll as the note on hits.

3.2 Use Case Modeling

This section describes the use cases of Rewind and covers the different scenarios of Rewind. The use cases were created to understand what the user needs are for Rewind. Both the back end of Rewind, being the models and web service, and the front end of Rewind, being the Graphical User Interface (GUI) of Rewind or the client, are covered by these use cases.

In the full use case diagram shown in Figure 8, there are four actors being the: User, Developer, Web Service, and the Rewind Server. The User are those who are interested in creating a transcription of a digital audio song. The Developer is one whom that is expanding and/or improving the accuracy of Rewind. The Web Service is a service that allows the Rewind client to convert a digital audio format into transcription. The Rewind Server serves a website to the Rewind client. The rest of the section explains each use case of Figure 8.

Play/Pause Playback
The user has the option to pause or playback a given transcription in the Rewind client.

Download Transcription
When a transcription has been received from the server, the user may download a transcription that one had requested.

Inspect Piano Roll
The user may look around the piano roll within the Rewind client.

Get Information About Project
The Rewind client will provide the user the option to get information about the Rewind project and how the project works.

Upload Audio File
The user in this use case will upload a file that they wish to transcribe.

Receive Transcription
When the server has received a transcription from the web service, the Rewind client will receive the transcription for playback and visualization.

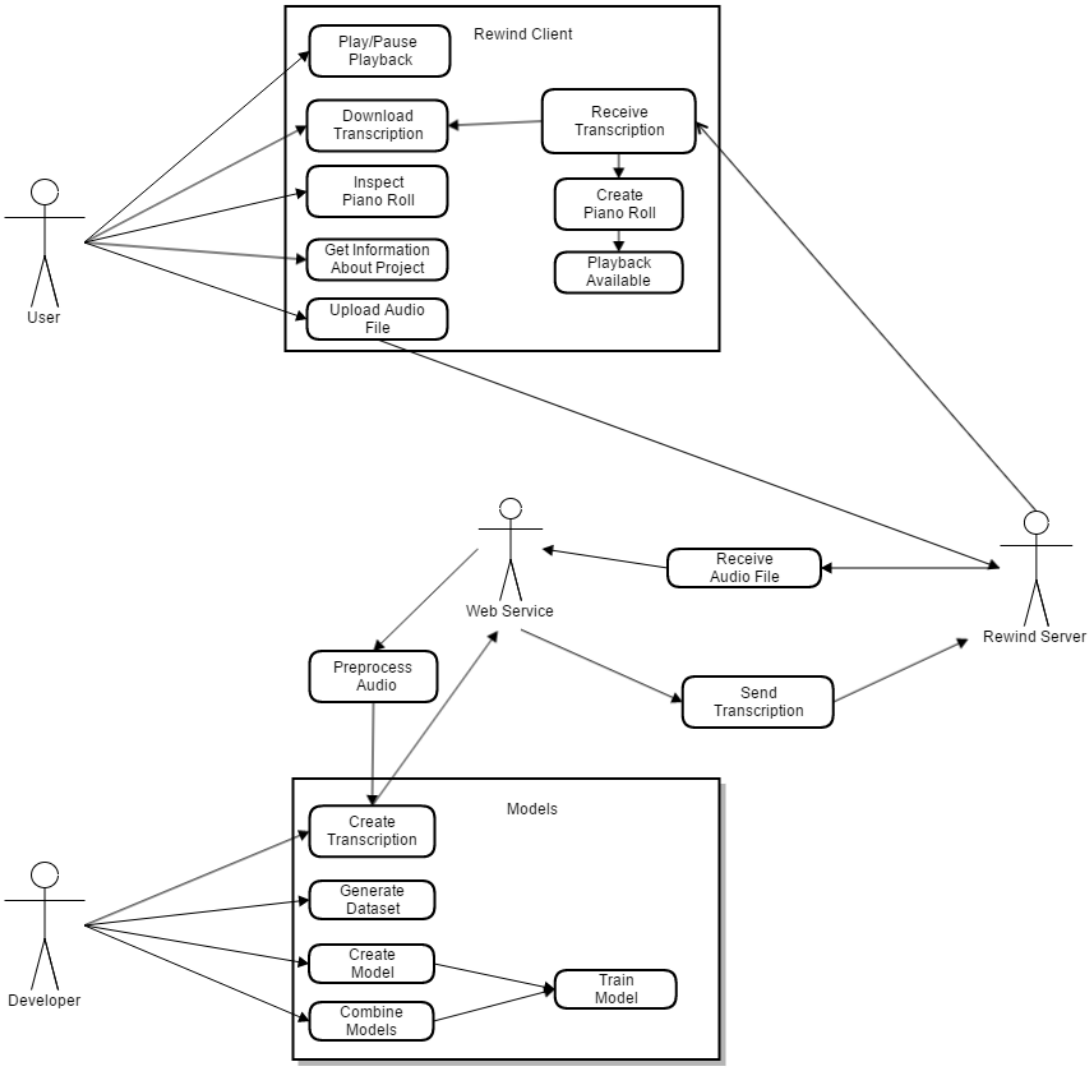


Figure 8: A Use Case Diagram of Rewind

Create Piano Roll

After receiving the transcription the rewind client will build a piano roll transcription for the user to see.

Playback Available

After the piano roll has been inside of the Rewind client, then the client will allow the user to playback the transcription and will let the user know that playback is available.

Receive Audio File

In this use case, the web service receives an audio from the server and is now ready to preprocess the audio file for transcription by the models.

Create Transcription

The create transcription use case can occur in two different ways: one is when the web service sends an audio file to the models for transcription or a developer invokes the service.

Send Transcription

When the models have finished transcribing, then the transcription will be sent to the web service where the Rewind server will then send the data to client.

Preprocess Audio

The models before they can transcribe any audio have to make sure that the files themselves are the proper format. If they are not proper, then by default the models will transform the music into the proper format.

Generate Dataset

The developer may wish to generate a new dataset for training the models, which is possible. This is so the developer may tweak Rewind and make its overall transcription accuracy better.

Create Model

The developer is also able to create new models that can be

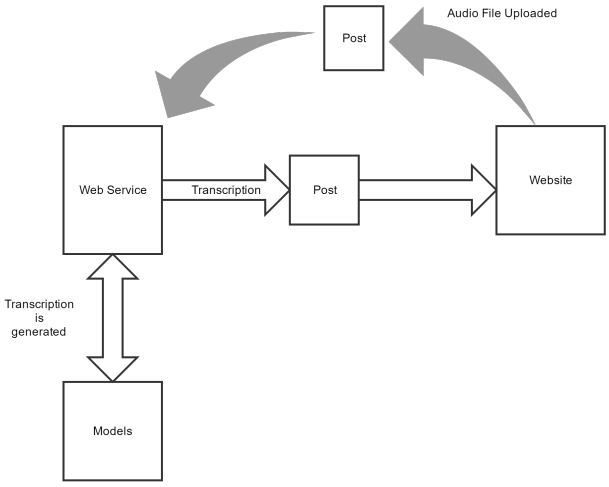


Figure 9: A diagram of Rewind's web service

utilized for transcription or research.

Combine Models

The developer may wish to combine multiple models together in order to improve transcription.

Train Models

The developer has the option of training the models in order to determine if the new model is better than the current model utilized by the web service.

3.3 Website and Web Service

Rewind's web service was implemented in Flask as a small web service that could be utilized by Rewind's server for making transcriptions of uploaded audio files. A small web service was implemented for transcribing audio files so that Rewind would remain scalable. All audio files and transcriptions are handled through post requests. Figure 9 demonstrates a diagram of the communication of audio files and transcriptions going in and out of the web service. This web service communicates with the models of Rewind and creates a midi file from the passed in audio file. All transcriptions generated by the web service are piano only. Rewind's website was implemented in the Django web framework and utilized the following javascript libraries: remodal, jQuery, jQuery UI, and midi.js. Django was chosen for Rewind because it allows Rewind to be scalable for future development, stable database integration, and future incorporation of security. Midi.js is utilized for its ability to parse MIDI files and generate sounds for those MIDI files. The jQuery and jQuery UI libraries has many useful features for designing interfaces, such as animations, element manipulation, and 3D effects. The remodal library allows for seamless modal windows to be displayed on the website. A small web service was implemented in Flask to wrap Rewind's models in order to be utilized by Rewind to generate transcriptions through http requests. This service was implemented so that the small web service could be independent and be used in other applications if needed. These libraries have made it possible to make a website for Rewind. An example of Rewind's website is demonstrated in Figure 6. This figure also demonstrates Rewind's ability to visualize the playback of a midi file in the form of a piano roll where the colors denote the note level. The user has the ability to scroll through the piano roll using the time bar and inspect the piano roll validity.

Rewind has a built in web synthesizer, which is used to playback transcriptions generated by Rewind's models. Midi.js has several dependencies, which are used to playback sounds and can handle different platform setups. It can parse midi events and make it possible to extract time delta for constructing piano rolls and note information. Midi.js can load many different sound fonts to load different sounds such as piano, flute, drums, and other sounds. The piano roll constructed for visualization in Rewind is based on the time duration and time position information collected from midi.js. The user has the ability to scroll through the piano roll using the time bar and inspect the piano roll validity. As a song plays the piano roll will light up each note with different colrs based on the note number as demonstrated in Figure 6, and the screen will transition to another part of the piano every second. There is some future work to be developed regarding the ability of adding or removing certain notes from the transcription using the piano roll. In conclusion, these libraries allow Rewind to be scalable for more complex models in the future.

4 Results

In this section we present the precision, recall, f-measure, and accuracy of Rewind's transcriptions on the following datasets: Nottingham consisting of 1000 or more songs, JSB Chorales consisting of 200 or more songs, Poliner-Ellis consisting of 30

songs, MuseData consisting of 700 songs, the Maps dataset consisting of 169 songs, and a custom dataset that consists of 160 songs split evenly from country, rock, jazz and classical. The custom dataset was added since all of the benchmark datasets currently used in the AMT are currently only classical piano music and orchestral music. All datasets are primarily midi and a synthesizer is used to generate wav except for the Poliner-Ellis and Maps dataset that have a aligned wav file and midi file. Rewind's model ran with two different models and both compared at a 10 ms and 50 ms stride. In Tables 1 and 2, the overall results of Rewind at a 10 ms stride, a standard for AMT systems, at the frame level are demonstrated and compared to Boulanger-Lewandowski's work [6, 24]. The 50 ms results are demonstrated in Table 3, but the results are not reported for the maps dataset. The 10 ms stride results were trained with two parallel GRUs with a linear layer and the 50 ms results were trained with a single GRU and linear layer. The results demonstrated in Table 2 are compared against ConvNet acoustic model at the frame level [24].

Upon examining the table, the Convnet is better overall in accuracy, recall, and f-measure, but Rewind has the higher precision. The ConvNet [24] utilizes a hash beam search to find the most probable sequence. If Rewind was to utilize the same hash beam search, it may have been able to achieve an even better accuracy, recall, and f-measure.

5 Conclusions and Future Work

Rewind demonstrated a encoder-decoder network that is comparable to the results of Boulanger-Lewandowski rnn-rbm [6] in terms of the Nottingham and JSB dataset. It also achieved a higher precision than the rnn-nade [24] on the Maps dataset. However, it suffered from issues in connection with choosing a threshold to generate an on value in the transcription on datasets such as MuseData and the custom dataset built by Rewind. The custom dataset demonstrated that AMT systems can work with multiple genres, but there may be other factors that cause transcription metrics to go down, such as multiple instruments being existent in the song or an improper threshold. Despite these issues, Rewind does manage to follow the underlying frame distribution in the lower classified datasets. Rewind's encoder-decoder has demonstrated a model that has a high precision and comparable results coupled with a web app that can generate transcriptions. Rewind's website provides users with a way to hear and see their transcriptions.

Rewind has demonstrated a model that works at the frame level. Previous work, such as [24], have used a frame level model in conjunction with a note level model to get a note level transcription. One key thing for the encoder-decoder network would be to add another layer, which can do note level transcription and utilize other algorithms from [6] to produce

Table 1: Rewind's results at 10 ms stride for the spectrogram (1 is the proposed model and 2 is the rnn-nade [6])

	Accuracy		Precision		Recall		F-Measure	
Models	1	2	1	2	1	2	1	2
Nottingham	95.1%	97.4%	98.0%		96.9%		97.5%	
JSB	82.8%	91.7%	92.4%		88.8%		90.6%	
Poliner-Ellis	34.4%	79.1%	66.9%		41.5%		34%	
MuseData	34%	66.6%	56.8%		45.9%		50.8%	
Custom	16.2%		51.1%		19.2%		27.9%	

Table 2: Rewind's performance on the Maps dataset compared to [24] at 10 ms.

	Proposed	Simple Auto-Correlation	ConvNet[24]
Accuracy	51.6%	6.4%	**58.87%**
Precision	**76.5%**	21.8%	72.40%
Recall	61.4%	8.2%	**76.50%**
F-Measure	68.1%	11.2%	**74.45%**

Table 3: Rewinds results at a 50 millisecond stride for the spectrogram where 2 is the proposed model and 1 is the Simple Auto-Correlation model

	Accuracy		Precision		Recall		F-Measure	
Models	1	2	1	2	1	2	1	2
Nottingham	21.5%	94.0%	29.2%	97.9%	44.7%	95.9%	35.3%	96.9%
JSB	20.8%	81.6%	32.9%	92.1%	36.2%	87.7%	34.5%	89.9%
MuseData	11.8%	23.0%	15.8%	60.2%	31.9%	27.2%	21.1%	37.4%
Poliner-Ellis	6.6%	42.6%	17.7%	70.5%	9.7%	51.8%	12.5%	55.8%
Custom	8.5%	20.4%	12.2%	44.5%	21.8%	27.3%	15.6%	33.9%

a more probable transcription. This is possible due to the separation of the encoder and decoder in the encoder-decoder network. Another encoder for Rewind could be designed for other problems such as genre classification, audio generation like [26], or audio transformation where a sound is transform into another sound. A deeper architecture could be considered for experimentation for the encoder network, using possibly more GRUs or LSTMs for larger datasets. One other issue that Rewind would like to solve is being able to produce a transcription for each instrument in a song and be able to determine what instrument is being played. Rewind's web has the potential for new features and interfaces for new problems. Rewind could be expanded into an application that allows a user to edit existing music that has been transcribed. Another addition would be to allow Rewind to recognize the lyrics of the music being played. One more thing that Rewind could provide is a way for users to collaborate and learn about music.

Acknowledgement

This material is based in part upon work supported by: The National Science Foundation under grant number(s) IIA-1329469. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

References

[1] James Allwright, *ABC Version of the Nottingham Music Database*, URL: http://abc.sourceforge.net/NMD/ (visited on 04/10/2016), 2003.

[2] James Allwright, *Bach Choral Harmony Data Set*, URL: http://archive.ics.uci.edu/ml/datasets/Bach+Choral+Harmony (visited on 04/10/2016), 2010.

[3] Mert Bay, Andreas F. Ehmann, and J. Stephen Downie, "Evaluation of Multiple-F0 Estimation and Tracking Systems," *Proceedings of the 10th International Society for Music Information Retrieval Conference*, http://ismir2009.ismir.net/proceedings/PS2-21.pdf, Kobe, Japan, pp. 315–320, 2009.

[4] Peter J Billam, *MIDI.lua*, URL: http://www.pjb.com.au/comp/lua/MIDI.html (visited on 04/10/2016), .

[5] Sebastian Böck and Markus Schedl, "Polyphonic Piano Note Transcription With Recurrent Neural Networks," *Acoustics, Speech and Signal Processing (ICASSP), 2012 IEEE International Conference on*, pp. 121–124, DOI: 10.1109/ICASSP.2012.6287832, 2012.

[6] Nicolas Boulanger-Lewandowski, Yoshua Bengio, and Pascal Vincent, "High-dimensional Sequence Transduction," *2013 IEEE International Conference on Acoustics, Speech and Signal Processing(ICASSP)*, pp. 3178–3182, DOI: 10.1109/ICASSP.2013.6638244, 2013.

[7] Nicolas Boulanger-Lewandowski, Jasha Droppo, Mike Seltzer, and Dong Yu, "Phone Sequence Modeling With Recurrent Neural Networks," *ICASSP*, IEEE SPS, URL: http://research.microsoft.com/apps/pubs/default.aspx?id=217321, 2014.

[8] Chase D. Carthen, "Rewind: A Music Transcription Method," MA thesis, University of Nevada, Reno, 2016.

[9] Chase Carthen, Vinh Le, Richard Kelley, Tomasz Kozubowski, and Frederick C. Harris Jr., "Rewind: A Transcription Method and Website," *Proceedings of the 25th International Conference on Software Engineering and Data Engineering (SEDE 2016)*, Denver, Colorado, USA, pp. 73–78, 2016.

[10] Center for Computer Assisted Research in the Humanities, *MuseData*, URL: http://musedata.stanford.edu/ (visited on 04/10/2016), 2016.

[11] Kyunghyun Cho, Aaron Courville, and Yoshua Bengio, *Describing Multimedia Content using Attention-based Encoder-Decoder Networks*, eprint: arXiv:1507.01053, 2015.

[12] Kyunghyun Cho, Bart van Merrienboer, Çaglar Gülçehre, Fethi Bougares, Holger Schwenk, and Yoshua Bengio, "Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation," *CoRR* abs/1406.1078 (), URL: http://arxiv.org/abs/1406.1078, 2014.

[13] Li Deng, Mike Seltzer, Dong Yu, Alex Acero, Abdelrahman Mohamed, and Geoff Hinton, "Binary Coding of Speech Spectrograms Using a Deep Auto-encoder," *Interspeech 2010*, International Speech Communication Association, URL: http://research.microsoft.com/apps/pubs/default.aspx?id=135405, 2010.

[14] Zhiyao Duan and Emmanouil Benetos, "Automatic Music Transcription," ISMIR, URL: http://c4dm.eecs.qmul.ac.uk/ismir15-amt-tutorial/, 2015.

[15] Valentin Emiya, *MAPS Database - A Piano Database For Multipitch Estimation And Automatic Transcription of Music*, URL: http://www.tsi.telecom-paristech.fr/aao/en/2010/07/08/maps-database-a-piano-database-for-multipitch-estimation-and-automatic-transcription-of-music/ (visited on 04/10/2016), 2008.

[16] Bernd Krueger, *Classical Piano MIDI Page*, URL: http://www.piano-midi.de/ (visited on 04/10/2016), 2007.

[17] Honglak Lee, Peter Pham, Yan Largman, and Andrew Y. Ng, "Unsupervised Feature Learning For Audio Classification Using Convolutional Deep Belief Networks," *Advances in Neural Information Processing Systems 22*, ed. by Y. Bengio, D. Schuurmans, J. Lafferty, C. Williams, and A. Culotta, pp. 1096–1104, URL: http://books.nips.cc/papers/files/nips22/NIPS2009_1171.pdf, 2009.

[18] Nicholas Leonard, Sagar Waghmare, Yang Wang, and Jin-Hwa Kim, *rnn : Recurrent Library for Torch*, eprint: arXiv:1511.07889, 2015.

[19] ofoct.com, *Convert WAV (or MP3, OGG, AAC, WMA) to MIDI*, URL: http://www.ofoct.com/audio-converter/convert-wav-or-mp3-ogg-aac-wma-to-midi.html (visited on 04/10/2016), 2016.

[20] Graham Poliner, *Automatic Piano Transcription*, URL: http://labrosa.ee.columbia.edu/projects/piano/ (visited on 04/10/2016), 2008.

[21] Armin Ronacher, *Django The Web Framework For Perfectionists With Deadlines*. URL: https://www.djangoproject.com/ (visited on 04/10/2016), 2016.

[22] Armin Ronacher, *Flask Web Development, One Drop At a Time*, URL: http://flask.pocoo.org/ (visited on 04/10/2016), 2016.

[23] Nicol N. Schraudolph and Terrence J. Sejnowski, "Unsupervised Discrimination of Clustered Data via Optimization of Binary Information Gain," *Advances in Neural Information Processing Systems*, Morgan Kaufmann, pp. 499–506, 1993.

[24] S. Sigtia, E. Benetos, and S. Dixon, "An End-to-End Neural Network for Polyphonic Piano Music Transcription," *ArXiv e-prints* (), arXiv: 1508.01774 [stat.ML], 2015.

[25] Siddharth Sigtia, Emmanouil Benetos, Srikanth Cherla, Tillman Weyde, Artur S. D'Avila Garcez, and Simon Dixon, "An RNN-Based Music Language Model for Improving Automatic Music Transcription," *15th International Society for Music Information Retrieval Conference (ISMIR 2014)*, 2014.

[26] Nitish Srivastava, Elman Mansimov, and Ruslan Salakhutdinov, *Unsupervised Learning of Video Representations using LSTMs*, eprint: arXiv:1502.04681, 2015.

Chase D. Carthen graduated from the University of Nevada, Reno with both a B.S. and a M.S. in Computer Science and Engineering in 2014 and 2016, respectively. He is currently working in industry as a software engineer. His research interests include human-computer interaction, graphics and simulations, and artificial intelligence.

Vinh Le graduated from the University, Reno with a B.S in Computer Science and Engineering in 2015. Vinh is a Graduate Research Assistant affiliated with the Cyber Infrastructure Lab at the University of Nevada, Reno. He currently aims to earn a Master of Science in Computer Science and Engineering by early 2017 and his research interests consist primarily of Software Engineering, Internet Architecture, and Human-Computer Interaction.

Richard Kelley is currently the chief engineer for the Nevada Advanced Autonomous Systems Innovation Center. He received his BS in Mathematics from the University of Washington in Seattle in 2006, and his MS and PhD in Computer Science and Engineering in 2009 and 2013 respectively. His research interests include robotics, human-robot interaction, machine learning, and unmanned aircraft systems.

Tomasz J. Kozubowski is a Professor in the Department of Mathematics & Statistics at the University of Nevada, Reno. He received his MS in Statistics from the University of Texas, El Paso in 1988 and PhD in Statistics and Applied Probability from University of California, Santa Barbara in 1992. He is a member of the American Statistical Association and the Institute of Mathematical Statistics, and works in the general area of probability and statistics. His research interests include theory of distributions, limit theory for random sums, heavy tailed distributions, extremes, mathematical statistics, financial and insurance mathematics, computational statistics, stochastic models for hydro-climatic phenomena, and fractal scaling processes.

Frederick C. Harris Jr. is currently a Professor in the Department of Computer Science and Engineering and the Director of the High Performance Computation and Visualization Lab and the Brain Computation Lab at the University of Nevada, Reno, USA. He received his BS and MS in Mathematics and Educational Administration from Bob Jones University in 1986 and 1988 respectively, his MS and Ph.D. in Computer Science from Clemson University in 1991 and 1994, respectively. He is a member of ACM (Senior Member), IEEE, and ISCA (Senior Member). His research interests are in parallel computation, computational neuroscience, computer graphics and virtual reality.

Rijndael Algorithm for Database Encryption on a Course Management System*

Francis Onodueze† and Sharad Sharma†
Bowie State University, Bowie, Maryland 20715, USA

Abstract

Effectiveness of any software system depends on techniques employed during access, storage, and retrieval. Securing a course management system with the latest security approaches is vital since it can contain information about students and faculty. Encryption is the most efficient way of securing data stored as it ensures that integrity of data is maintained even if an attacker should gain access to the physical data in the database. Encryption can occur at different levels starting from data, disk to the entire device. This paper presents the implementation of Rijndael Algorithm for a database encryption on a Course Management System, to provide an additional level of security to the information of students, faculty and overall data of the software. We have also provided benefits and drawbacks of various database encryptions based on the amount of data encrypted and te modes of access to keep a balance between efficiency and security. Furthermore, we apply these techniques on a web interface which uses Microsoft technologies to accept users' login details, goes though encryption process, and stores cipher text in the database.

Key Words: Encryption, cipher, database, cryptography, Microsoft dot net.

1 Introduction

Ability of the computer to perform more functions creates the need for more data to be stored. When a computer had less power, it did not store much information because it could be compromised. Only passwords were considered secure data and the computer took as much time to encrypt and decrypt it. However, we have seen the computer's ability increase with proven strength that it can secure more than just passwords. Today, data retrieved from users range from login details to personal data of individuals which they would not share with any human. The confidence users have come from the fact that once their information is encrypted and stored, not even the system admin can retrieve its plan text data. Encryption is a very efficient way of securing data, it helps ensure data confidentiality and integrity in different communication systems, data storage and networks [11]. As defined in [3], encryption algorithms consist of complex mathematical formulas that define the rules of conversion process from plain text to cipher text and vice versa combined with a key. Encryption is achieved using the technique of Cryptography, which is a science that learns the mathematical techniques of keeping information secured [25]. Cryptography converts the original message into unreadable codes and makes sure the original message cannot be retrieved except by reverse process using an appropriate key [7].

Encryption algorithms can come in two forms, public or private Encryption keys, depending on the specifics of each service, application and volume of data to be secured. Public key encryption is a cryptographic system that makes use of pairs of keys. While one key if disseminated publicly, the other is known only to the party that decrypts the message. Amongst the widely known public encryption algorithms is RSA which is a short form for Rivest-Shamir-Adleman who were the developers of the algorithm [18]. Private key encryption is a cryptographic system that uses the same key for encryption and decryption. The cryptographic key used in a symmetric algorithm is often transferred over a secured channel and kept secret by both parties. Some of the private encryption algorithms, also called symmetric algorithms include Data Encryption Standard (DES). Triple DES (TDES), which was derived from encrypting DES three times and Advanced Encryption Standard (AES) which is a standard specification for electronic data. Encryption algorithms can also be classified based on the size of data encrypted in each encryption cycle and size of key. Encryption algorithms are designed to use different length of keys, from 56-bits up to 256-bits, the more the key length, the more secured would be the algorithm and the more resistant it would be for brute force attack [6].

Encryption keys must have two basic attributes to be determined secured; key space and random selection. The key space is determined by the key length and composed of all possible permutations of the keys. Key spaces are designed to make almost impossible for an attacker to search through the set of all possible keys. Random selection determines keys are chosen randomly from all possible keys. Otherwise, an attacker can derive some similar factor that may determine how the key selection was done. Brute force takes the encrypted file and checks all possible combinations of generated keys until a match is found [17]. Most attackers first try dictionary attack before

*Extended paper from proceedings of ISCA 25th International Conference on Software Engineering and Data Engineering (SEDE-2016), Marriott Tech Center, Denver, USA, September 26-28, 2016.

†Department of Computer Science. Email: onoduezef0108 @students.bowiestate.edu, ssharma@bowiestate.edu.

brute force because it often produces correct answers in less time than a systematic brute force attack. A dictionary attack tries all possible string in pre-arranged manner, to determine the decryption key or passphrase. Whereas, brute force searches all possible character combination systematically in no given order. Brute force attack is often used to measure the strength of encryption algorithms because it is the highest known form of successful attack. Other types of attack include: mathematical analysis which falls in the same category of classical attacks as brute force; and social engineering which occurs when someone uses his closeness to an actual user to gain unauthorized access.

There are many other forms of attack but since brute force is an algorithm based on chance, it is difficult to determine the probability of its success. However, the probability of success can be significantly reduced by increasing possible combinations in the search space. This could be done by increasing the key length and possibly the encryption rounds. The longer the encryption key, the more the robustness of the algorithms against brute force attack [1]. Some of the present day's algorithms were developed when no one thought computers would possess the power they have today. Therefore, already known algorithms with proven strength can be made to go through several re-encryptions to increase their strength. For instance, the DES(x) algorithm was re-encrypted in the form of DES(DES(DES(x))) to make the algorithm stronger and more resistant to attacks. Triple DES takes 3 times the processing time of DES and implements DES algorithm with an extended key length [24]. Figure 1 shows a typical cryptology structure with categories and algorithms. We have followed this principle in designing our application and have made possible for the administrator to be able to extend the key length used for encrypting data in this application.

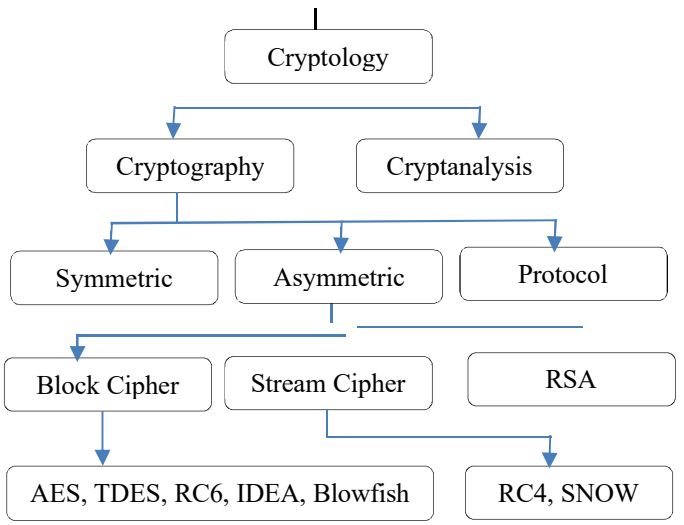


Figure 1: Cryptology structure

1.1 Encryption Technique

Encryption techniques are a way of transforming physical form of data, so its confidentiality, integrity, authentication and validity is maintained. These techniques are applied in such a way that even though a hacker gains access to the data, he would not be able to read the information contained. The goal of encryption is to make data difficult to use or access by unauthorized persons, however, the science behind can basically be understood as manipulating individual characters of the string to be encrypted. This is usually not done manually or by easily substituting individual characters. Although, when encryption first started, obfuscating data only meant substituting individual characters of the plain text string with characters that would make the entire text un-meaningful. However, technology increases and approaches continue to differ. Once strong algorithms became easy to break and thus caused the need for a more secured approach. This has been the story since conception of the first known algorithm. Today, encryption involves several complex computations and substitution based on giving algorithms. Modern cryptography requires a mathematical approach to help understand and teach the subject and also to improve on existing algorithms. Encryption algorithms can be classified into two:

- Public-key or asymmetric algorithms (e.g. RSA)
- Private-key or symmetric algorithms (DES, TDES, AES).

Private-key cryptosystem deals with a mechanism which requires that the key be made private and shared securely. They are more efficient and work with special protocols and shared keys for key exchange and management.

No one algorithm claims to be 100% efficient in securing data and information from hacking attempts but a strong encryption could be a good way to increased security.

Our proposed course management system incorporates encryption for the purpose of increased security on the information of students, faculty and the entire university system. Information contained in the university academic database could include student and faculty personal information, social security numbers, student grades, faculty study materials and personal research contents. However, the major gateway to accessing this information is through access control mechanisms such as user authentication using username, email, and password or phone number. This is the information which can be encrypted. Other information is not included because the intruder needs this authentication information in other to gain access to every other data.

To encrypt input data, we run each plain text password through a software program, to convert it into an unreadable format. This process could take a lot of processor time and memory if used to encrypt every information in the database. Several approaches were discussed which can reduce the issue of time and speed on database encryption.

This paper applies encryption to the data that controls login access to the user account. The course management application was designed to handle academic interaction between professors, researchers, teaching assistants and students in a social environment. It can be used by professors to send and receive assignments from students, post students' grade and

manage students' record. Even though the application was built to increase study and improve lecturer-student communication, we have built in social media features. We included a social feel to the application because almost every student interacts perfectly with social media and would not have trouble navigating through the functionalities of the application. The application provides academic features to include: upload and download of assignments and class materials, maintain a personalized e-book shelve for users, classroom forum, discussion board and messaging among registered users. It also includes social features such as posting comments to wall, update your status, reply to posts, invite friends, create circles and Like what-you-see. As the user's would have to provide personal information to use the software, it became imperative we ensured an efficient encryption algorithm was used to secure data of this software. This paper focuses on how the data of the course management system is encrypted and does not go into details on other functionalities provided by the course management system.

2 Related Work

2.1 Block Cipher

Block cipher is a terminology used to define the pattern of input fed into the encryption cycle in a given time. It is the contrast of stream cipher. Stream ciphers encrypt input bits individually by adding a bit from a key stream to a plaintext bit thereby changing the original value. Each bit is encrypted by XOR'ing the plaintext with the state [27] to ensure every bit is changed when one bit is changed. Stream cipher is small and fast. It is used on cell phones, embedded devices and on applications that have little computational resources. Block cipher encrypts an entire block of plaintext bit. Each block is encrypted with the same key at a time to ensure the influence of each bit is spread across all bits. Block ciphers are used widely both in software and in hardware implementations. They are used in electronic payments and for wireless security. For different demands, different algorithms are designed [13]. Apart from this main functionality, block ciphers are also used as underlying primitives in the design of hash functions or pseudo-random number generators [22]. In software implementation, it cannot be physically seen whether a given algorithm is block or stream cipher. This is because the algorithm is written as a class which is called by the program or written as part of the program. What determines a given implementation is the underlying data structure, data manipulations, computation and resource usage such as memory and CPU. In our program, we encapsulate the encryption codes and make them accessible via method calls. While this approach makes for a neater code, it also adds additional level of security to the application.

Our proposed course management system uses Rijndael encryption algorithm for password encryption. Rijdndael algorithm is a block cipher designed by Daemen and Rijmen [9]. In Rijndael, both the block length and key length can be any multiple of 32 bits, with a minimum of 128 bits and a maximum of 256 bits; independent of each other with key size greater than or equal to the block size. Since the algorithm can accept varying key and block lengths, the 128-bit block variant of Rijndael has been chosen as the standard for Advanced Encryption Standard (AES) [6].

2.2 Rijndael Algorithm

Rijndael algorithm was chosen by the National Institute of Standards and Technology (NIST), USA in 1998 as the best algorithm for the Advanced Encryption Standard (AES). The algorithm was selected in a process that involved five finalists that were selected from an original 21 submissions. Since after this selection process, it has been widely accepted and used in many cryptography applications, and although the length used by the algorithm was too small, it was embraced and has been used by different government agencies, companies and organizations.

Rijndael supports key sizes of 128, 192 and 256 bits [12, 14, 21], with data handled in blocks of 128 bits. It has two major parts of transformation; key schedule and cipher rounds. Key schedule is an iterative component used to make the cipher resistant to attacks. The cipher rounds ensure that each bit in the block depends on all other bits of the state. It has become the most widely used symmetric encryption and industry standard for many commercial systems. Among companies that have adopted this cipher method are: IPsec, TLS, IEEE 802.11i, SSH (Secure Shell), Skype and many other companies where security is of utmost importance. The internal structure of AES consists of the following stages:

- byte substitution layer
- diffusion layer
- key addition layer
- key schedule

Encryption: A 16-byte input data is fed byte-wise into the S-Box (A0, ..., A15), the S-box transformation also is called the byte replace (SubBytes) [6]. The Byte Substitution Layer consists of 16 parallel S-Boxes which forms a one-to-one objective mapping with the input elements.

Diffusion ensures that the influence of individual bits is spread over the entire state. Diffusion consists of the *ShiftRows* transformation and the *MixColumn* transformation. *ShiftRow* transformation is a cyclical shift in the state matrix. The rows are shifted to the right according to the row positions. The second row is shifted by three bits, the third row by two bits, the fourth row by one byte and the first remains unshifted. The row shifting in AES is used to increase diffusion. MixColumn transformation is a linear mix of the columns in the state matrix.

AES ensures that after the diffusion process of three rounds, every byte of the state matrix depends on the entire 16 plaintext bytes. The Key Addition Layer takes two 16-byte (128 bits) inputs, which are the state matrix and a subkey. The two inputs are XORed together [26]. Key Schedule is a transformation

technique used to derive the subkeys used in AES. Choosing an appropriate key schedule has been found to successfully improve the time complexity of differential attacks on AES 128-bit [20]. It uses the master (secret) key as an input to generate round keys (subkeys). The master (input) key length in bits can be 128, 192 or 156 [6]. In [14], it states that the minimum input block for AES is 128-bit. AES key schedule is word-oriented, where 1 word = 32 bits.

AES is one of the most well-studied block ciphers. It forms the basis for a well-accepted crypto-system which can be used to demonstrate the internal structure of a well secured algorithm. It has undergone several analyses and most importantly withstood the efforts of brute force attacks. Some other forms of attack used to prove its strength include: boomerang attacks, square attacks, impossible differential attacks, rectangle attacks and man-in-the-middle attacks in both the single-key and related-key settings [15, 23]. Rijndael decryption is the reverse process of its encryption, following the exact same order.

2.3 Implementation Overview

The course management software helps professors manage students' course work and grades. It has features that allows users (students, professors and researchers) to communicate. It is a social application that has functionalities that can be used in the education sector. The application was built to accommodate different institutions while ensuring that personal or universities' specific data does not get intermixed. Upon first usage, users are required to sign up using the name of their institution, create a user name and password, and select if he/she is a student, research/teaching assistant or professor. In subsequent sign-in, users will be asked to provide their registered email and password to be launched into their personal university profiles. Encryption is done at both registration and login process using database column encryption as discussed in section 2.4. During registration, while user details are captured, the password is encrypted and saved along with other data. During every login, user provided login password is encrypted and matched against saved encrypted passwords to determine if the user should be granted access or not.

The software provides the following services to the users:

1. All users should be able to sign into his/her account using the correct user name and password.
2. All users should be able to upload and download files/assignments.
3. All users should be able to send messages to other users.
4. All users should be able to search for and find other users and participating institutions.
5. All users should be able to update his/her status.
6. All users should be able to reply to other users' posts.
7. All users should be able to create circles and add other users to the circles.
8. Professors should be able to add materials to their shelves.
9. All users should be able to invite other people to use the software. Invite is sent using either email address or phone number.

2.4 Database Encryption

As mentioned in Section 2.3, not every data in the database is encrypted. Encryption can be very expensive on devices with less capability. In this application, we encrypt the password field while leaving the other fields in plain text. Although this is less costly, it is a valid database encryption technique. This section shows various database encryption techniques and its impact on overall system performance.

On database encryption, what comes to mind is column encryption. Many times there are no specifics as to what exactly is encrypted. However, database encryption can occur in three levels as shown in Figure 2.

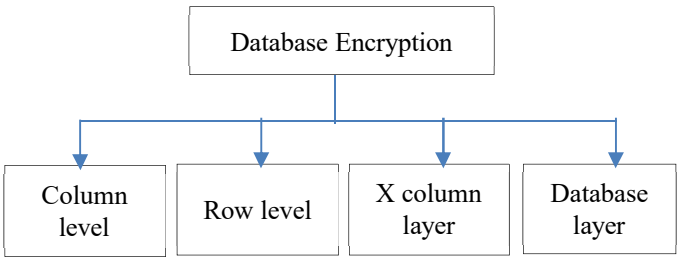


Figure 2: Level of database encryption

2.4.1 Column Level. In column level encryption, one or more columns are encrypted in the database as shown in Figure 3. Encryption and authentication are done at application level where inputs are received from the user. This method of encryption is targeted at securing the most important items in the database, such as usernames, passwords, social security numbers, credit card details and any sensitive data. It is less time consuming and device efficient since not all data is encrypted. The fewer the sensitive data, the less work of encryption during storing and retrieval of records.

id	username	password	department	ssn
123	John	xxxxxxx	HR	xxxxxxx
456	Philip	xxxxxxx	ACCT	xxxxxxx
789	Joan	xxxxxxx	HR	xxxxxxx

Figure 3: Column level database encryption

2.4.2 Row Level. This is used when the database is small and contains mostly sensitive information. It ensures that all data in the database is encrypted. This can cause a major challenge on a large database if encryption and decryption is done at the application level. Some database management systems provide this service and thus ease the workload on the application during information retrieval. Figure 4 shows row level encryption.

2.4.3 Extreme Column Level. Extreme column level (XCL) is a technique whereby different columns in the database is encrypted using different encryption keys, as shown in Figure 5.

id	username	password	department	ssn
123	xxxxxxx	xxxxxxx	xxxxxxx	xxxxxxx
456	xxxxxxx	xxxxxxx	xxxxxxx	xxxxxxx
789	xxxxxxx	xxxxxxx	xxxxxxx	xxxxxxx

Figure 4: Row level database encryption

XCL is efficient because it makes life tougher for a hacker. Although this method is more efficient than column and row level encryption, it comes with a lot of performance overhead. It needs adequate key maintenance since a lot of keys might be involved in encrypting a simple database. Encryption and decryption is resource and time consuming because each item in the database goes through a different routine before it can be deciphered into readable text.

	username		password	department	ssn
key	xyx		Pqy	ngk	kjd
data	xxxxxxx		xxxxxxx	xxxxxxx	xxxxxxx

Figure 5: Extreme Column Encryption on single row

2.4.4 Database Level. The below three levels of encryption can be described as application level encryption because encryption and decryption is done on application level. The database holds the cipher texts. Database Level Encryption (DLE) occurs when the entire database framework is encrypted. Most organizations rely on this level of database encryption as it does not require extra application level encryption. It can be risky not to have an additional level of encryption because the records in the database are stored as plain text, which means bypassing this level of encryption exposes the records in the database. However, accessing is fast as applications do not need to do any encryption or decryption work.

DLE usually serves as additional security with an existing application level encryption. DLE is more than passing a string in a method call for encryption as done in application level encryption. Therefore, it is provided as a service to organizations by database service providers [4] or sold as a software to users. The concept is similar to encrypting a document, CD or drive, which are services provided by several companies or computer system manufacturers such as Microsoft and Apple. Companies such as Oracle and NetLib provide database encryption services and software. In SQL server, a database needs to be attached to an instance of an active server in order to be accessed. Therefore, databases can exist outside of a server. Databases are detachable and pluggable, hence the term pluggable database (PDB). When encrypted and detached, it can exist alone and safely carried along.

When a database is detached from its origin, it can be vulnerable if access was restricted only at the server level. When DLE is used, it prevents the database from being attached to unauthorized instances of other servers. Attempts from attackers, through the operating system, to read data from the tablespace or backup is denied. This protects databases on backup media from unauthorized network, domain and windows administrators. It secures databases from SQL *sysadmin* and provides authorized users a dedicated SQL instance for the database. Intellectual properties of the database such as, schema, views, stored procedures, tools and other business processes are also secured using this database wide encryption.

However, there are lots of overheads in using this encryption technique, such as Single Key Entry (SKE), which is a risk when there is no other level of encryption on the database. SKE means, when an intruder gets a hold of this key, he would have unlimited access to every item contained therein. Another drawback of this method is that it does not protect data travelling over the network. As the name goes, it does not mean the time spent on encrypting a database is comparable in quantity to encrypting the entire text data, on the application level. Database encryption makes use of high speed cryptographic techniques designed to overcome performance overhead. DLE uses the same concept as Transparent Data Encryption. Access to the data is transparent which means authenticated users and application are not blocked away from the encrypted database. Providing encryption to database using DLE does not require any change to the database configuration or application code.

3 Course Management Architecture

The application was designed on Microsoft Dot Net platform, and uses combined languages such as VB and C# for its coding. In Microsoft.net, dynamic program coding is handled using code-behind model. This is a model that allows the developer to place his codes in separate file or a specifically designed script tag. Code-behind files use the same name as the page and has vb. extension. It was introduced to give developers the ability to separate presentation from business tier and eliminate classic ASP (Active server pages) which strictly ties codes to their pages.

The course management application was built on a three-tier architecture namely: Presentation tier, Business tier and Data Access tier. The presentation tier consists of html and ASP tagged contents rendered as .aspx and .ascx (user control pages) on the browser. These pages interact with the user and provide interfaces for collecting login details and display of texts and graphic content. The presentation layer does not do any form of processing nor connect to the database. Its values are extracted to the code-behind files in the business layer and processed. Other elements in the presentation layer include Ajax, JQuery, CSS and Inline JavaScripts.

As shown in Figure 6, the code-behind model is the second layer in the presentation tier. It consists of files which have direct access to the presentation layer. These files are responsible for receiving authentication details entered by the user such as username, email and password. Each time a user attempts to login, this file takes care of validating user inputs to make sure they don't violate set criteria. The retrieved login credentials are passed to the encryption business logic for encryption and decryption.

The Business tier consist of class files, Web services (JSON),

ADO.Net system packages, Dot Net Objects and COM objects that perform actual operations on the page. When data is passed from the presentation, it is given to the business for encryption, key generation, calls to the Dynamic Link Library (.dll) and connection to the Data access tier. Unlike the presentation layer which is client-side, the business is server-side, its codes are not loaded directly on the user's end.

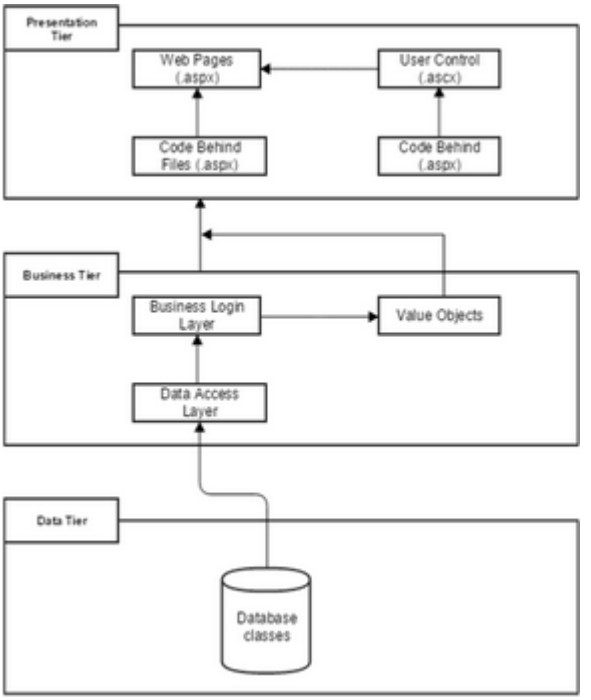


Figure 6: Dot net design architecture using code behind model on a three-tier architecture

This model helps ensure that direct access to the encryption codes is not possible. The code-behind files do not have direct access to the database because they can give a hacker access to the database by SQLInjection; especially if calls to the database are inline SQL statements. Best practices suggest that if database calls must be made in the presentation, they should be via stored procedures. It is highly discouraged to have any database code in the presentation layer. Every information storage and retrieval has to be by invoking the data access classes.

The Data access tier consist of codes that make direct connection to the database for creation of database schemas, insert, update and delete of tables and other manipulation using queries and stored procedures. These help to pull user details whenever the user tries to login in. The course management application uses SQL Server for data storage and implements column level database encryption. It uses Internet Information Server (IIS) as its application host. Figure 7 shows a class diagram for the login, registration and their relationship to other classes in the application. The application host is Internet Information Server (IIS).

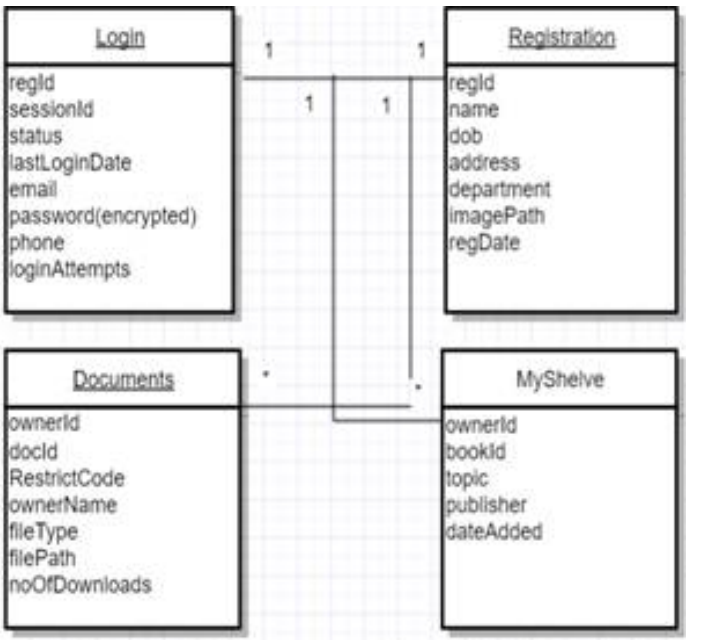


Figure 7: Class Diagram

4 Implementation of Rijndael Algorithm on Course Management System

Our proposed Course Management System was built on the Dot Net platform using Microsoft Dot Net controls and objects. The pages which serve as views render as ASPX pages. Each page contains a code-behind file which serves as the controller and contains codes that interface between the data classes and views/presentation. Codes which do not have a direct link to the views are stored on class files to make sure the code is not compromised in case an intruder gets access to the website files. This technique is used to increase security as these class files would be pre-compiled in object codes and stores in .dll formats which would be unreadable should the software be hacked. Encryption and decryption codes are stored in class files amongst code files that perform different functions such as SendEmail, DataBaseConnection, KeyGenerator CheckInternetConnectivity etc.

Our encryption and decryption codes are stored in Encryption.cs. This is a C# dot net class file which contains the encryption functions which later is called to perform the actual encryption. This class is bundled as part of the project namespace to ensure the functions would be available on all pages. When the encrypt function is called, an unreadable value (cipher text) is returned and that would be the data stored in the database, as Figure 3 shows. This encrypted string can also be exposed in the URL as query strings without fear of exposing the data contained. Such techniques can ensure that the database administrator may have access to the database data and still not be able to see the plaintext data of the encrypted texts, in this case the login details of the registered users.

Figure 8, shows what happens during a typical authentication process. The presentation layer is where login details are collected, the encryption stage, the flow of events and the storage of the cipher text in the database. The user enters his username or email and password and clicks on the register button to commence registration.

For the purpose of demonstration not all data in the database is encrypted. And in real-life application, not all data in the database is encrypted. Encryption is costly in terms of the process involved in actual encryption and decryption of stored data. During the process of encryption or decryption, a large amount of memory and CPU is used due to complex repetitive functions and computation performed. That can make the computer slow and unresponsive if we were to encrypt all data in the database. When data is encrypted, the resulting data would be alphanumeric character string. The sequence of character representation in the encrypted data is strictly determined by the algorithm.

During registration, the user password is passed to the Encrypt function of the Encryption class as strings, and the returned value is stored in the database.

At login, when the user enters his login details, the Encrypt function is called as shown in Figure 3. It is used to wrap the password, thus passing the string to the Encrypt function of the Encryption class. A string is returned which is used as part of the WHERE condition of the SQL statement. This statement is then executed against the SQL server engine via a method call to the DBConnection class. If the returned encrypted strings match any value in the database, the Boolean value receives a true value. Otherwise a false value is returned and the user sees an invalid login attempt message on the screen and prompted to retry.

Figure 8 shows database security access layers [2] for which every access to the database must belong. For each of these layers, a rule is defined in the database and in the application. These rules help enforce strict security regulations and check for policy compliance. These rules are enforced to ensure users comply to measures that help secure their data. When a user is authenticated, a session is saved to the database to monitor an active browsing session of the user. The login details are persisted on browser cookies and used for validation throughout this session.

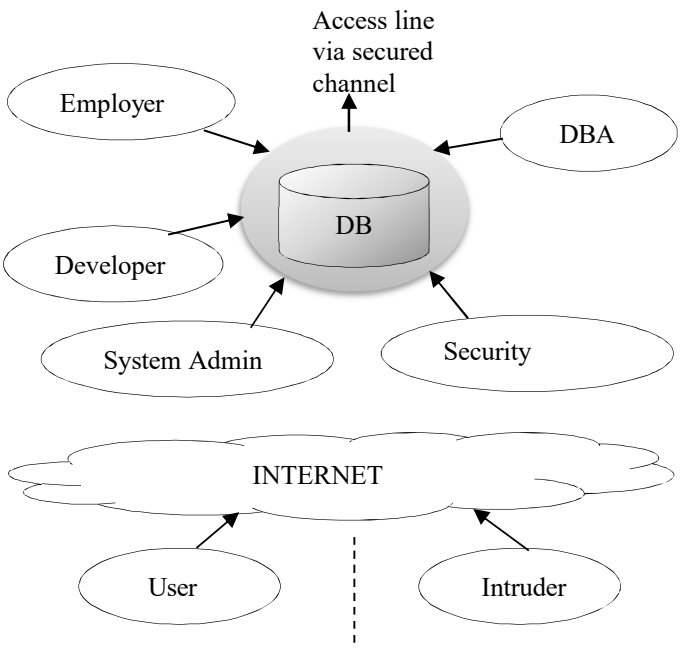


Figure 8: Database security access layers

This is done to ensure the user is validated on every page he accesses and no page is opened by unauthorized persons. The encrypted information is stored on varchar (225) datatype in the Microsoft (MS) SQL database. User login details saved on the browser are passed as strings into the application for user validation to recognize the user anytime he accesses any page. These cookies are set to expire with the session. When the user session expires, the cookies will no longer be available in the browser and in the database. The user is then redirected to the login page

5 Result and Testing

The Encryption.cs class is implemented using C#.Net code, while the actual call from the web page is in VB.Net. This is made possible by the ability of the Dot Net platform to compile several languages into object code. However, for this to work, the separate class files have to be placed in different folders. We have placed our Encryption.cs class file in a folder called *App_code*, our web page will still remain in the project's folder. To use the Encryption.cs class, we instantiate it into a *rijndael* object. This object is then used to call Encrypt and Decrypt functions for the login page processing.

We have showed how Microsoft applications can be secured using Rijdael encryption algorithm to encrypt login authentication details and how cipher data can be stored in the database more efficiently.

5.1 Encrypting the Login details

The login details are passed to the Encrypt function as strings to the *Encrypt* function. This function takes a plain string named *clearText* and passes it to the function body for processing. The function is made up of the following as shown in Figure 9:

1. *EncryptionKey* variable where the key is declared
2. *clearBytes* array which stores individual data bytes as they are encrypted
3. *using* statement which references the Cryptography library of the Microsoft.Net framework
4. *return* statement which returns the cipher text back to the calling statement.

The calling program is an SQL *Insert* statement which formats its input into an appropriate insert statement. It makes a call to the *DatabaseConnection.cs* class and executes the statement against the database.

Since the Encrypt function is called in an insert statement, we do not expect any error to be thrown once the format of the statement is verified. Once the insert is successful, the user is navigated to the next screen to complete his registration process.

```
public encrypt(plain text)
{
string encryption key
convert plain text to byte
using (create AES encryptor
{
  pass the key to the encryptor
  pass plain text data byte to encryptor
  using (create memory stream
 {
   using (encrypt plain text)
   {
     write cipher text to memory stream
     close memory stream
   }
   return cipher text
} }
}
```

Figure 9: The Encrypt function

5.2 Decrypting the Login details

On the user login page, information entered in the textboxes is captured as strings and passed to the *rijndael* object of the Encryption.cs C# class. The *Decrypt* function contains a key string which must be the same as that of the Encrypt function. Otherwise, decryption will not be successful. The process of decryption is similar as the codes show above but the difference is that, this time, the string passed is a *cipherText* not a plain text.

Figure 10 shows a database view of the session id, user email and password. The session ids with *null* values show that the user is no longer on active browser session. The password shown is cipher text generated by the *Encrypt* function of our Encryption.vb class. Column level encryption has been used on this project.

SESSION_ID	EMAIL	PASSWORD
in3igd5caybbi0jml0i2tqvo	paul@yahoo.com	zHqtPVrr7bJJEjWUhFeuUA==
in3igd5caybbi0jml0i2tqvo	paul@gmail.com	zHqtPVrr7bJJEjWUhFeuUA==
null	francis.onodueze@gmail.com	x8V2lQdiKY5VMpb+hIdrMw==
null	joan@gmail.com	2wQyJPbywulbjcLuSpAYhw==
ma3w0lvj4prjkyh4wzxrj1vr	kyndra@gmail.com	2wQyJPbywulbjcLuSpAYhw==

Figure 10: SQL Server database view shows the encrypted data

6 Conclusion and Future Work

We have implemented Rijndael private key algorithm to show how security can be increased in a course management system using encryption. There is no doubt that adding encryption to the existing course management system has improved dramatically the confidentiality and integrity of information of students and faculty.

The encryption was implemented using a static code as the encryption key. The algorithm will generate the same cipher text each time for the same input value. This means a hacker who gets this key can decrypt the values stored in the database. This method could be safer if the encryption key is a securely kept secret. However, this is not the safest way. To increase the protection level of this algorithm, the encryption algorithm could be connected to a random number generator to generate password salts. This would make it possible for the algorithm to generate different cipher values for the same plain text during encryption and decryption.

Acknowledgements

This work is funded in part by the National Science Foundation grant number HRD-1238784.

References

[1] A Study of Methods used to Improve Encryption Algorithms Robustness, Luminita Scripcariu, Faculty of Electronics, Telecommunications and Information Technology, Technical University "Gheorghe Asachi" of Iasi, ROMANIA. pp. 1-5, 2015.

[2] Khaleel Ahmad, Jayant Shekhar, Nitesh Kumar, and K. P. Yadav, "Policy Levels Concerning Database Security," *International Journal of Computer Science & Emerging Technologies (E-ISSN: 2044-6004) 368*, 2(3):368-372, June 2011.

[3] O. M. A. Al-Hazaimeh, "A New Approach for Complex Encrypting and Decrypting Data, *International Journal of Computer Networks and Communication (IJCNC)*, 5(2):95-103, March 2013.

[4] I. Bashrat, F. Azam, A. W. Muzaffar; "Database Security and Ecryption: A Survey Study," *International Journal of Computer Applications* (0975-888), 47(12):28-34, June 2012.

[5] J. Daemen, L. R. Knudsen, V. Rijmen, 'The Block Cipher Square', FSE, (LNCS 1267), pp. 149-165, 1997.

[6] J. Daemen and V. Rijmen, *The Design of Rijndael: AES – The Advanced Encryption Standard*, Springer, 2002.

[7] D. Darin and E. Harley, "Selling E-Learning," *American Society for Training and Development,* pp. 1-10, 2001.

[8] D. S. Elminaam, H. M. Abdual Kader, and M. M. Hadhoud, "Evaluating the Performance of Symmetric Encryption Algorithms," *International Journal of Network Security*, 10(3):213-219, May 2010.

[9] FIPS, 197: *Advanced Encryption Standard*, Federal Information Processing Standards Publication 197, U.S Department of Commerce/N.I.S. T, 2001.

[10] Anwar Pasha Abdul GafoorDeshmukh; "Transparent Data Encryption-Solution for Security of Database Contents," *(IJACSA) International Journal of Advanced Computer*

Science and Applications, 2(3):25-28, March 2011.
[11] J. Katz and Y. Lindell, *Introduction to Modern Cryptography*, Second Edition, CRC Press, 2015.
[12] Publication 800-21, *Guideline for Implementing Cryptography in the Federal Government*, National Institute of Standards and Technology, November 1999.
[13] National Bureau of Standards: *Data Encryption Standard,* FIPS-Pub.46. National Bureau of Standards, U.S. Department of Commerce, Washington DC, January 1977.
[14] J. Nechvatal, E. Barker, L. Bassham, W. Burr, M. Dworkin, J. Foti, and E. Roback, *Report on the Development of the Advanced Encryption Standard (AES)*, Technical Report, NIST, 2000.
[15] Christof Paar and Jan Pelzi, *Understanding Cryptography*, Springer-Verlag, Berlin, Heidelberg, ISBN: 978-3-642-04100-6, pp 115-121, 2010.
[16] R. C.-W. Phan, "Impossible Differential Cryptanalysis of 7-Round Advanced Encryption Standard (AES)," *Information Processing Letters*, 91:33-38, 2004.
[17] B. D. Reddy, V. V. Kumari, and KVSVN Raju, "A New Symmetric Probabilistic Encryption Scheme Based on Random Numbers," 2014 First International Conference on Networks & Soft Computing (ICNSC2014), Guntur, pp. 267-272, 2014.
[18] R. L. Rivest, A. Shamir, and L. Adleman, "A Method for Obtaining Digital Signatures and Public-Key Cryptosystems," *ACM*, 21(2):120-126, 1978.
[19] P. Srinivasarao, P. V. Lakshmipriya, P. C. S. Azad, T. Alekhya, K. Raghavendrarao, and K. Kishore, "A Technique for Data Encryption and Decryption," *International Journal of Future Generation Communication and Networking*, 7(2):117-126, 2014.
[20] S. Sulaiman, Z. Muda, and J. Juremi, "The New Approach of Rijndael Key Schedule," *Proceedings of the 2012 International Conference on Cyber Security, Cyber Warfare and Digital Forensic (Cyber Sec)*, Kuala Lumpur, pp 23-25, 2012.
[21] M. Wali and M. Rehan, "Effective Coding and Performance Evaluation of the Rijndael Algorithm (AES)", *Proceedings of the Engineering Sciences and Technology Conference*, Karachi, 7:1-7, 2005.
[22] Qingju Wang, Zhinqiang Liu, Deniz Toz, Kerem Varem Varici, and Dawu Gu, "Related-key Rectangle Cryptanalysis of Rijndael-160 and Rijndael-192," IET Information Security, 9(5):266-276, August 2015.
[23] Z. Wentao, W. Wu, and F. Dengguo, *New results on Impossible Differential Cryptanalysis of Reduced AES in Information Security and Cryptology*, ICISC 2007, Lecture Notes in Computer Science, Springer Berlin / Heidelberg, 4817:239-250, 2007.
[24] Zhou Yingbing and LI Yongzhen, "The Design and Implementation of a Symmetric Encryption Algorithm Based on DES," IEEE 5th International Conference on Software Engineering and Service Science, Beijing, pp 517-520, 2014.
[25] Zahir Zainuddin and Evanita V. Manullang, "E-Learning Concept Design of Rijndael Encryption Process," Proceedigs of IEEE International Conference on Teaching, Assessment and Learning for Engineering (TALE), Bali, pp. 737-740, 2013.
[26] F. Zhang and Y. Niu, "Rijndael Arithmetic Analyze and Optimize," 4th International Conference on Wireless Communications, Networking and Moile Computing, Dalian, pp. 1-4, 2008.
[27] Y. Zhang, J. Sun and X. Zhang, "A Stream Cipher Algorithm Based on Conventional Encryption Techniques," Canadian Conference on Electrical and Computer Engineering 2001 (IEEE Cat. No. 04CH37513), pp. 649-652, 2004.

Francis Onodueze was born in Nigeria. He received the N.D degree in computer science from Federal Polytechnic Nekede, Imo State, Nigeria, in 2004, B.Sc. degrees in computer science from the Imo State University, Imo State, Nigeria, in 2009, and the M.Sc. degrees in computer science from the Bowie State University (BSU), Maryland, USA, in 2016.

Francis is a Ph.D. student at Bowie State University and serves as a teaching/research assistant. His current research interests include artificial intelligence, software security and software engineering.

Sharad Sharma is an Associate Professor in the Department of Computer Science at the Bowie State University. He received a Ph.D in Computer Engineering from Wayne State University, Detroit, MI in 2006 and M.S. from the University of Michigan, Ann Arbor, MI in 2003. He has won the "Outstanding Researcher Award" in year 2013 and 2011, "Outstanding Faculty Award" in year 2012, "Outstanding Publication Award" in year 2010, and "Outstanding Young Faculty Award" in year 2009 at College of Arts and Science in the Bowie State University. Dr. Sharma is the Director of the Virtual Reality Laboratory at the Bowie State University. The laboratory applies virtual reality and augmented reality as a tool for learning, training, and education. Dr. Sharma's research focus is on modeling and simulation of multi-agent systems for emergency response and decision making strategies. His work is motivated by the need of research in real-time agent navigation for reaching a goal in emergency situations like evacuation.

Evolution of the Multicore Adaptability of Scientific Software Systems

Saleh M. Alnaeli*, Melissa M. Sarnowski*,
Calvin Meier*, and Mark Hall*
University of Wisconsin, Fox Valley, Wisconsin USA

Abstract

An empirical study that examines the challenges that scientific software systems developed in C/C++ have that prevent them from efficiently exploiting the full potential of the new multicore technology is presented. The study is conducted on 12 open source scientific systems comprising more than 5.4 million lines of code and containing over 84.5 thousand for-loop statements. Static analysis methods are applied to each system to determine the number of for-loops and free-loops (i.e., loops that can be parallelized). Additionally, each system is analyzed and the challenges and inhibitors to parallelization from a software engineering perspective are detected and presented. Some challenges towards adapting and re-engineering scientific software systems to better utilize modern multi-core architectures are determined including function side effects, data dependency and jumping statements. The results show that the most prevalent inhibitors are functions called within for-loops that have side effects, followed by data dependency as the second most prevalent inhibitor. These inhibitors pose the greatest roadblock to re-engineer and transform systems to better utilize parallelization. Results also show that data dependency has a more significant impact on scientific systems compared to general purpose systems that have been studied in previous studies. Historical data over a 5-year period of inhibitor counts for the set of systems studied is also presented. It shows that there is an insignificant change in the potential for parallelization of for-loops over time in general. The study suggests some software engineering techniques that have the potential to improve the parallelizability of scientific systems.

Key Words: **S**cientific, software engineering, multicore architecture, parallelization inhibitors, challenges.

1 Introduction

Scientific software systems have always been valuable assets and are used by scientists in both academia and industry in their research for many purposes (e.g., analyze and solve research and scientific problems, or to interpret, visualize, or simulate processes and data). A great deal of those systems are available as open source for research communities from various scientific and engineering disciplines.

With new high performance computing architectures (e.g., multicore technology) becoming increasingly available in almost all of today's computers, laptops, and mobile devices, there is a continuous need for the existing scientific application code to adapt in order to exploit and take advantage of the full potential of the new underlying hardware. In contrast to the general-purpose software systems, where programmers and software engineers do not usually possess expertise in the problem domains of the software they develop, people who have reasonable knowledge of the system domain (e.g., physics, mathematics, geology, engineering, astronomy) usually develop scientific software systems or assist with the development. Those scientific programmers are more likely to have a background lacking some of the most important software engineering methods and concepts that allow programmers to design and write programs that can be easily adapted to outside changes, such as hardware improvement.

For instance, scientific programmers typically make inefficient use of the shared memory multicore technology versus applications focused on distributed memory architectures. The problem gets worse as the number of cores increases in shared memory models, which may reduce individual core speed and cause drastic slowing in sequential software speed. As a result, many end users, some of which are scientific developers, are starting to feel that they need to ensure that the scientific application they use has an acceptable performance when run on various devices (e.g., tablets, laptops, or mobile devices with multi-core architecture). The expectation that various devices are used has motivated scientific developers to refactor their products to enable parallel execution and take advantage of underlying hardware [2].

The process of parallelizing a software system usually involves the use of standard APIs such as OpenMP. These APIs provide the developer with a set of tools to parallelize loops and take advantage of multiple cores and shared memory [2]. Current C/C++ compilers can do a limited amount of automatic parallelization. That is, the compiler directly parallelizes loops with fixed iteration bounds (i.e., for-loops) in certain situations. Loops without fixed iteration bounds cannot, in general, be parallelized. The auto-parallelization can also be done via a tool prior to compiling. These tools look for four-loops that do not contain any parallelization inhibitors [5, 11].

*Email: saleh.alnaeli@uwc.edu, sarnm6825@students.uwc.edu. meiec3837@students.uwc.edu, mark.hall@uwc.edu.

For scientific software systems, many obstacles can pose challenges for parallelization. Those challenges vary in their significance and severity. Some difficulties are caused by the way developers write their source code [9]. However, in the end, the full potential of multicore processor architectures will require a deeper assessment of the scientific application to adapt the source code for scalable concurrency and better parallelizability.

As both scientists and researchers, we share the same concerns and opinion that these older scientific systems need to be refactored to take advantage of parallelization. Software designers need to understand these concerns so that scientific software systems can exploit the full potential of the new high performance computing systems (e.g., multicore architectures) that are available in almost every computer and device today [11].

This study takes an empirical approach to understand the complexities for these scientific open source software systems. This approach allows software developers to better understand the challenges and roadblocks inhibiting parallelization of these systems. We are particularly interested in determining the most prevalent inhibitors that occur in these scientific applications and any general trends. This work serves as a foundation for understanding the problem requirements in the context of a broad set of scientific applications. Moreover, the focus of this research is on potential challenges and the most common situations occurring within typical scientific software systems. The types of software challenges are counted and tabulated for a comparison across these types of systems. The data in the table allows a comparison of these systems, which helped uncover software trends and to make other general observations. This research is necessary to understand the challenges that these systems face and the impact and significance of overcoming these challenges in order to have practical and effective solutions that can lead to overall system parallelizability. Finally, a historical analysis is conducted to see if the numbers or distributions of inhibitors change over the history of a software system. Trends are shown in a later section.

This work contributes in several ways. First, it is one of the only large studies on the potential to parallelize scientific software systems. Our findings show that function calls with side effects and data dependency represent the vast majority of inhibitors occurring in these systems. That is somewhat contradictory to the case with general-purpose software systems [2], where function calls with side effects represent the vast majority of inhibitors and thus pose the greatest roadblock to adapt and re-engineer general-purpose systems to better utilize parallelization. This knowledge and fact will assist researchers in formulating and directing their work to address those problems for better multicore-capable scientific systems in a different way than how general-purpose systems are treated.

The rest of the paper is broken into sections. Section 2 presents related work on the topic of scientific systems performance and parallelization. Sections 3 and 4 describe the methodology we used in the study along with how we performed the analysis to identify each inhibitor and side effect. Section 5 presents the data collection processes. Section 6 presents the findings of our study of 12 open source scientific systems, followed by Sections 7 and 8 which discusses historical analysis.

2 Background

The bulk of previous research on this topic has focused on the importance of scientific systems and their quality from different perspectives [9], and detecting and dealing with data dependencies, particularly in the context of array indices [2], even though there are many other inhibitors that appear to be more frequent as we are going to show [8]. However, no study has been conducted to show the actual challenges in scientific systems to be able to utilize the advantages of multicore architectures on the source code level, nor from a software engineering perspective.

In our study, implicit parallelism is considered with the shared memory parallel model [4, 5]. There are multiple APIs used for parallel programming (e.g., MPI, PThread, OpenMP). Our concern in this study is to support the parallelization of existing sequential code. OpenMP is the most common API and our discussions are within the context of using this API. It is a widely accepted standard and most of the compilers support it [12, 13]. OpenMP is a set of standards and interfaces for parallelizing programs in a shared memory environment. It provides a set of pragmas (C/C++) that can be used in a program to instruct compilers to parallelize pieces of code. The sequential code is incrementally parallelized and the program can have both serial and parallel code.

There is a large body of work on parallelization. In the 1960s, parallel computers and research on parallel languages, compilers, first began. The focus was instruction-level parallelism [10] and mainly involved detecting instructions in a program that could be executed in concurrent to reduce the computation time.

Parallelizing compilers, such as Intel's [11] and gcc [7], have the ability to analyze loops to determine if they can be safely executed in parallel on multicore systems, multi-processor computers, clusters, MPPs, and grids. The main limitation is effectively analyzing the loops. For example, compilers still cannot determine the thread-safety of a loop containing external function calls because it does not know whether the function call has side effects that would introduce dependences.

S. Alnaeli, J. Maletic et al [2] conducted an empirical study that examines the potential to parallelize large-scale general-purpose software systems. They found that the greatest inhibitor to automated parallelization of for-loops is the presence of function calls with side effects and they empirically proved that this is a common trend. They recommended that more attention needs to be placed on dealing with function call inhibitors, caused by function side effects, if a large amount of parallelization is to occur in general purpose software systems so they can take better advantage of modern multicore hardware.

The work presented here differs from previous work on scientific software parallelization in that we conduct an

empirical study of actual inhibitors to parallelization in the source code level. We empirically examine a number of systems to determine what roadblocks exist to developing better parallelizable scientific software systems that can better work on multicore architecture.

3 Inhibitors to Parallelization

We now discuss the potential challenges programmers need to address when writing or refactoring scientific software systems for parallelization in the source level code. Inhibitors to parallelization are discussed along with the kinds that are known to be solvable by OpenMP.

In this study, a for-loop is considered a free-loop if it does not contain any parallelization inhibitors that are not already solvable with OpenMP. That is, a free-loop does not contain any of the following inhibitors: data dependency, function calls with side effects, or jumps outside of the loop.

This section mainly describes the different inhibitors to the software parallelization process and the challenges that prevent scientific systems from running in parallel for better multicore exploitation. Particularly, we are interested in for-loop parallelization inhibitors because, in most applications, the extensive computation is carried out in loops and parallelization APIs, like OpenMP, can parallelize only for-loops. However, not all for-loops are parallelizable. For example, for-loops whose results are used by other iterations of the same loop will not work properly, and can lead to unexpected and incorrect results. Inhibitors can prevent for-loop parallelization. While some of these are solvable, others are not.

Some inhibitors have a direct solution in Application Programming Interfaces such as OpenMP, and others cannot be solved and demand more complex (conservative) approaches. In this study, a for-loop is considered a free-loop if it does not contain any parallelization inhibitors that are not already solvable with OpenMP.

The data dependency is discussed first, followed by function calls with side effects, and then jump statements (e.g., break, goto).

3.1 Data Dependency

Data Dependency is a well-studied problem in many different contexts, including software slicing and static analysis, that inhibits software systems from parallelization. In many situations, the order of statement execution within the body of the for-loop must be preserved to gain the same expected results from a software system, as when executed in sequential order. That is, all loop iterations must be independent from each other.

Literature is rich when it comes to data dependency tests. Most of those tests have been developed based on approximation. All methods are conservative in case of dependency suspension, or when it is difficult to prove the opposite, so that no unsafe parallel implementation is done.

The main purpose of data dependency analysis is to detect if the same memory position is used in more than one loop iteration. The majority of dependency analysis algorithms are focused on array references. There are three types of dependency based on the way and the sequence of accessing a memory location. They are 1) flow dependence (aka true dependence), 2) anti-dependence, and 3) output dependence. This topic is well covered in [3].

Figure 1, presents a simple example that shows the data dependency in a loop construct. In the example, if the loop is parallelized evenly on 2 cores with 2 threads (500 iterations each), when the element 501 is calculated it would result in a wrong answer because the items 499 and 500 have not been calculated yet if the second thread acquires the CPU before the first thread. That is, items that come before the current item need to be calculated first.

```
Example 1: Fibonacci sequence:
// 0, 1, 1, 2, 3, 5, 8, 13, 21,34, …
1: array[1]=0;
2: array[2]=1;
3: for(int idx=3; idx<=10000; ++idx)
4: array[idx] =array[idx-1] + array[idx-2];
```

Figure 1: Example of data dependency detected by the tool ParaStat tool

In this work, we take a conservative approach to detecting data dependency and detect all potential dependencies. That is, if we cannot prove that a loop is free of dependency, we consider it a potential data dependency holder.

We feel this is a reasonable tradeoff since only simple static analysis is required. Also, there are always situations where determining if an actual data dependency exists is computationally impractical.

3.2 Function Calls with Side Effects

Another challenge scientific software systems in particular, and general-purpose software systems in general, can have in the parallelization context is calling functions or methods that have side effects within a for-loop. Today's compilers cannot parallelize any loop containing a call to a function, or a routine, that has side effects. A side effect can be produced by a function call in several ways, all related to any modification of the nonlocal environment, such as modification of a global variable, or passing arguments by reference [8]. Moreover, a function call in a for-loop or in a call from that function can introduce data dependency that might be hidden [14]. The static analysis of the body of the function increases compilation time; hence this is to be avoided.

As such, it is usually left to the programmer to ensure that no function calls with side effects are used and the loop is parallelized by explicit markup using an API. Generally, a function has a side effect due to one or more of the following: the function

modifies a global variable; modifies a static variable; modifies a parameter passed by reference; performs I/O; or calls another function that has side effects. Our approach for calls using function pointers and virtual methods is to assume they all carry side effects. At the onset, this may appear to be a problematic, however conservative, limitation. However, this assumption is supported by empirical analysis we undertook in a previous study [1].

3.3 Jumps: Break, Goto

Breaks and goto statements are inhibitors to the parallelization of for-loops. That is, the loop must be a basic block, meaning no jumps outside the loop are permitted. As such, the occurrence of one of these statements prevents parallelization of the loop. It is very simple to detect all occurrences of break and goto statements in source code so counting them is accurate. A call to exit() can be handled by OpenMP, so we do not consider these as loop inhibitors. Also, the same applies to exception handling. Exceptions thrown in a parallel region and caught within the same region are safe for parallelization. Catches can be inserted into those regions automatically if they do not exist. Since there is a known solution for exceptions, we do not consider them as inhibitors.

3.4 Shared and Private Data

There are some other inhibitors that can prevent the for-loop from parallelization. Shared data and private data are both inhibitors that must be taken care of. In this study, we do not consider them since we are assuming that the OpenMP API is used for loop parallelization. OpenMP offers particular directives that can solve these problems. Variables that are shared among all threads can be problematic because if one thread is reading it, another thread may be writing to it [2]. OpenMP can solve this problem using a special directive for that:

#pragma omp parallel for private (sharedVar);

Reduction variables can also be inhibitors. However, OpenMP has a special directive that solves this problem:

#pragma omp parallel for reduction (SharedVar);

This clause makes the reduction variable shared to generate the correct results, but private to avoid race conditions from parallel execution. Since those types of inhibitors are solvable by OpenMP, they are not considered in our study and we consider the loops containing these inhibitors to be free-loops.

4 Methodology of Inhibitors' Detection

A for-loop is considered a free-loop if it does not contain any parallelization inhibitors that are not already solvable with OpenMP. We used a tool, ParaStat, developed by one of the main authors and used in [2], to analyze loops and determine if they contain any inhibitors as defined in this section. First, we collected all files with C/C++ source code extensions (i.e., c, cc, cpp, cxx, h, and hpp). Then we used the srcML (www.srcML.org) toolkit [6] to parse and analyze each file. The srcML format wraps the statements and structures of the source code syntax with XML elements, allowing tools, such as ParaStat, to use XML APIs to locate such things as for-loops and to analyze expressions. Once in the srcML format, ParaStat iteratively found each for-loop and then analyzed the expressions in the for-loop to find the different inhibitors. A count of each inhibitor per loop was recorded. It also recorded the number of free-loops found. The final output is a report of the number of free-loops and for-loops with one or more types of inhibitors. Findings are discussed later in this paper along with limitations of our approach.

5 Data Collection

Software tools were used, which automatically analyze loops to determine if they contain any inhibitors. The srcML toolkit produces an XML representation of the parse tree for the C/C++ systems we examined. ParaStat analyzes the produced srcML produced using XML tools to search the parse tree information using system.xml from the .NET framework. The body of each loop and function is then extracted and examined for each type of inhibitor in loops or side effects in functions.

For the loops, if no inhibitors exist in a for-loop, it is counted as a free-loop; otherwise, the existence of each inhibitor is recorded. The systems that were chosen in this study were carefully selected to represent a variety of scientific systems developed in C/C++. These are well known scientific systems used by large-scale research and governmental institutions e.g., NASA, and many other systems that are known to communities from both academia and industry.

6 Discussion

We now study the challenges in parallelizability of twelve open-source scientific systems, including inhibitors to parallelization and function side effects. Table 1 presents the list of scientific systems examined along with number of files, and LOCs for each system. Table 1 shows a count of how many for-loops were found in each system and for comparison the number of while-loops and number of functions in each scientific system. One item of interest is that all of the scientific software systems show a much larger use of for-loops than while-loops. This is promising for potential parallelization through the use of APIs such as OpenMP.

6.1 Design of the Empirical Study

Our study focuses on four aspects regarding for-loops in scientific software systems. First, the percentage of for-loops containing one or more inhibitors gives an indication of how much the system could be readily parallelized by a compiler or other automated tool. Second, we examine which inhibitors are the most prevalent. Third, we seek to understand when

Table 1: The 12 open source scientific systems in the study, and functions, loops found in the 12 systems. All systems were updated to last revision in 2016

System	KLOC	Funs	For	While	Files
3DSlicer	4,029	**11,366**	**2,577**	**377**	**2,008**
Growler	1,591	**4,307**	**391**	**324**	**274**
Gwyddion	1,185	**8924**	**4,476**	**506**	**631**
Madagascar	922	**11,623**	**37,859**	**531**	**3,544**
NightShade	888	**4,247**	**449**	**185**	**239**
PSI4	736	**10,422**	**24,777**	**405**	**1,992**
RtRetrieval	695	**84,380**	**8,143**	**5,624**	**1,128**
Waffles	565	**8,164**	**3916**	**800**	**568**
Cantor	503	**1,491**	**81**	**56**	**522**
Fityk	449	**1,843**	**661**	**101**	**379**
Step	391	**1,190**	**160**	**9**	**245**
Marble	2,356	**9,294**	**1,091**	**333**	**4,936**
TOTAL	**5,417,408**	**157,251**	**84,581**	**9,251**	**16,466**

inhibitors are the sole cause in preventing parallelization. That is, loops can have multiple inhibitors preventing parallelization and therefore, would require a large amount of effort to remove all the inhibitors. Because of this, we are interested in understanding how often only one type of inhibitor occurs in a loop. These types of loops would hopefully be easier to refactor into something that is parallelizable. We propose the following research questions as a more formal definition of the study:

RQ1: What is a typical percentage of for-loops that are free-loops (have no inhibitors)?

RQ2: Which types of inhibitors are the most prevalent?

RQ3: Is there a higher percentage of for-loops that are considered free-loops as a scientific system evolves over time?

and **RQ4**: Have the most prevalent inhibitors of a system changed over time?

We now examine our findings within the context of these research questions.

6.2 Percentage of Free for-loops

Tables 1 and 2 present the results collected for the 12 scientific systems. We give the total number of free for-loops along with the total number of all for-loops we detected.

Table 1 shows the percentage of free-loops computed over the total number of for-loops. As can be seen, free-loops account for between 48% and 81% of all for-loops in these systems, with an overall average of 66%.

Table 2: Parallelization inhibitors of the 12 scientific software systems used in the study (last revision in 2016)

System	Function Calls with Side Effect	Jumps (goto and breaks)	Data Dependency	Free for-loops
3DSlicer	5.70%	5.39%	8.57%	81.68%
Growler	9.46%	10.74%	12.53%	69.05%
Gwyddion	27.65%	8.51%	13.47%	56.56%
Madagascar	24.70%	2.21%	40.45%	48.05%
NightShade	14.92%	14.92%	8.24%	68.15%
PSI4	20.56%	1.63%	29.89%	53.77%
RtRetrieval	18.50%	11.62%	0.17%	72.70%
Waffles	22.19%	7.30%	13.45%	62.64%
Cantor	14.82%	13.58%	0%	74.07%
Fityk	15.28%	7.26%	13.31%	68.53%
Step	15.00%	6.88%	7.50%	73.75%
Marble	24.20%	10.72%	4.40%	63.34%
TOTAL	**18%**	**8%**	**13%**	**66%**

However, in general, the percentage is high for most of the scientific systems compared to the situation in general-purpose systems and other domains (e.g., middleware) that were studied in our previous study e.g., [2].

That is, on average, a big portion of the detected for-loops in these scientific systems could potentially be parallelized. This addresses RQ1.

6.3 Parallelization Inhibitors

We now use our finding to address RQ. Table 2 presents the details of our findings on the distribution of inhibitors in studied scientific systems.

It clearly counts each of the inhibitors that occur within for-loops. Many of the for-loops have multiple inhibitors (e.g., a data dependency and a jump). As can be seen, function-call inhibitors are the most prevalent across most of the systems. However, in two systems data dependency was the primary issue. For most of the systems, this is followed by data dependency, and then jumps, thus addressing RQ2. All jump, goto and break, code were grouped together. This is similar to our findings with general purpose systems, though scientific systems show better results by a noticeable margin [2]. The findings show that 3DSlicer has great potential to be parallelized and take advantage of multicore challenges than the others when it comes to the parallelization context.

Figure 2, presents the average percentage, over all 12 scientific systems, of for-loops that are not free and contain at least a single inhibitor. This gives a clear view of which inhibitor occurs most frequently. We see that the trend is likewise similar via this perspective. Function-call inhibitors are the most prevalent with about 5% higher than data dependency, which is not the case with general-purpose software systems previously studied. On average, we see the next most prevalent inhibitor is data dependency, followed by jumps.

7 Historical Analysis

A set of 8 systems from the original 12 were chosen, all of which have been under development and maintenance for at least 5 years in order to address questions RQ3 and RQ4. The other systems were either relatively new and thus have a short history, or their history was not accessible. Therefore, they were excluded from this historical analysis. The goal is to uncover how each system evolves in the context of multicore adaptability.

7.1 Evolution of Free For-Loops

Figure 3 shows the percentage of free-loops for each year's version of each of the 8 scientific systems studied. Even when reduced to 8 open source scientific software systems, the overall percent average of free-loops remains the same at 66%. The overall average percent of free-loops across the 2012 version of all 8 systems is about 67%. As the systems grew over a 5-year period, the average percent of free-loops remained similar or had a very small increase or decrease. This shows us that the developers of open source scientific systems are not making significant improvements towards the parallelizability of their systems. This addresses RQ3. The findings show that there is not a higher percentage of free-loops in the scientific systems as they evolve over time, but rather a decrease in the percentage of free-loops over the 5-year period studied.

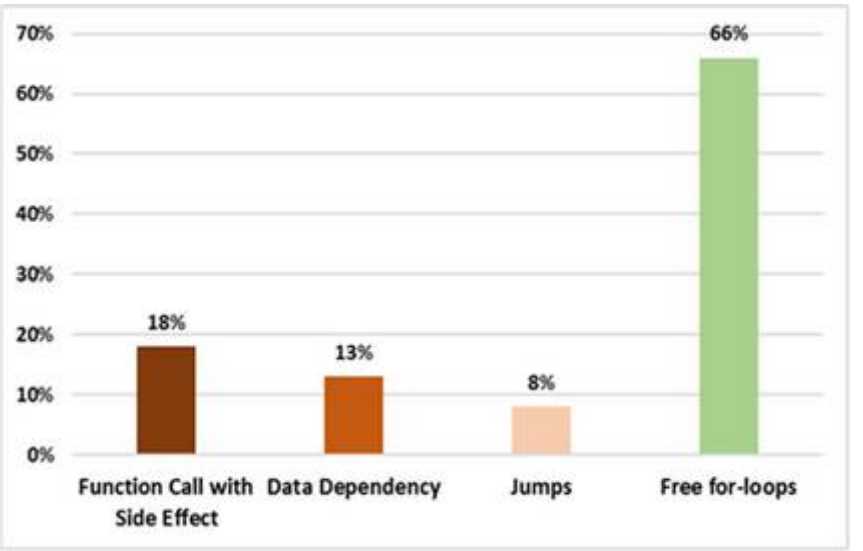


Figure 2: The average percentage of free For-Loops and Non-Free-For-Loops that contain at least one inhibitor, over all 12 systems.

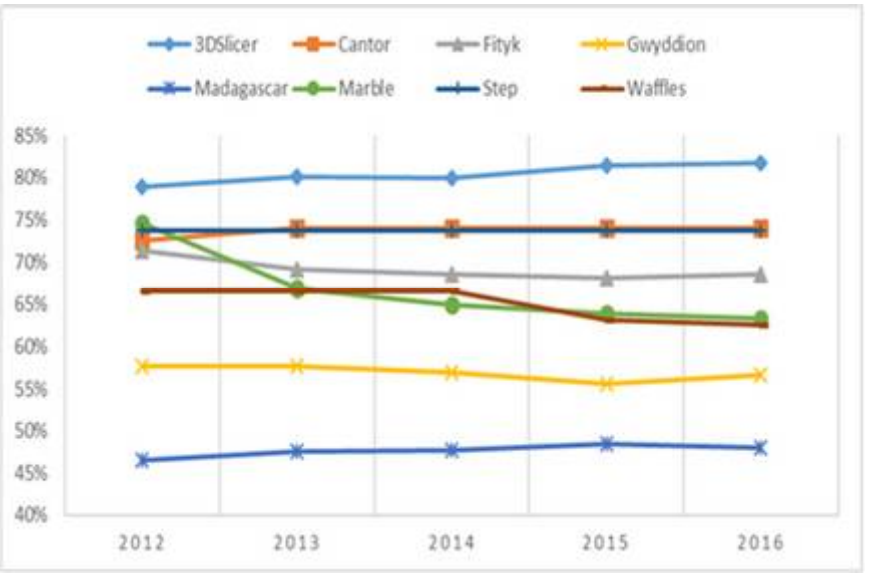


Figure 3: Percentage of Free-loops for each of the 5 studied years of each system

7.2 Inhibitor Prevalence

In each of the 8 studied open source scientific software systems, the inhibitor that was the most prevalent in 2012 was the same inhibitor that was the most prevalent in 2016, which was data dependency or function calls with side effects for most of the studied systems. In fact, for most systems, the number of each inhibitor stayed relatively similar or increased from 2012 to 2016. This addresses RQ4, as the most prevalent inhibitor for each system remained the same over the 5-year period.

7.3 Uncovered Historical Trends

The results of this study show that open source scientific systems are, in general, not becoming more parallelizable as they evolve over time. Figure 3 shows the percentage of free-loops for each version of the systems. Most systems show fairly flat trends, but 3DSlicer shows a slight increase in percentage of for-loops that are free-loops, while Marble and Waffles show decreases. Even if the percentage of free for-loops remained relatively the same, the percentage of jump statements, data dependencies, and direct calls to functions with side effects tended to increase or remained similar throughout the 5-year period studied for most systems.

Figure 4 shows the percentage of for-loops with direct calls to side effects over the 5-year period. Once again, most systems show a flat trend, but Marble and Waffles show an increasing trend that correlates to their drop-in percentage of free-loops. Figure 5 shows the percentage of for-loops with data dependency in the systems. Madagascar maintains a high

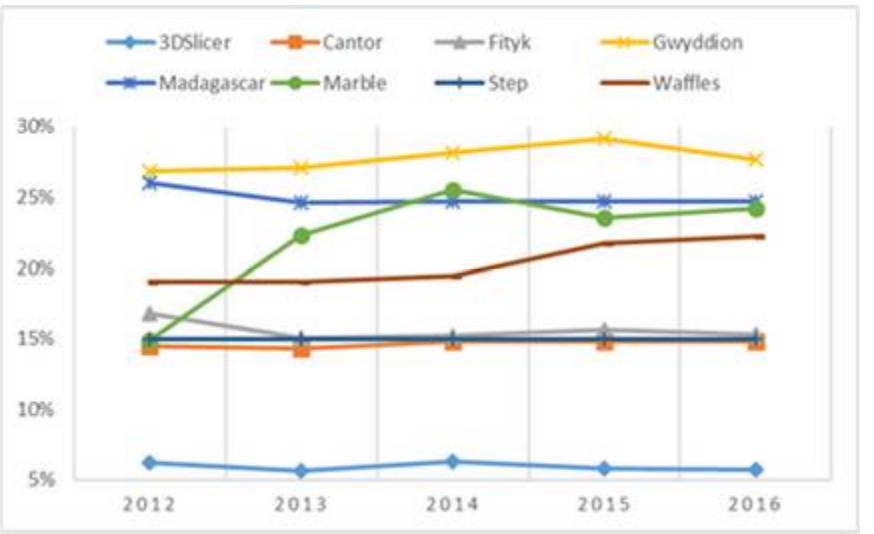


Figure 4: Percentage of For-Loops with a direct call to a function with side effects in all studied scientific systems

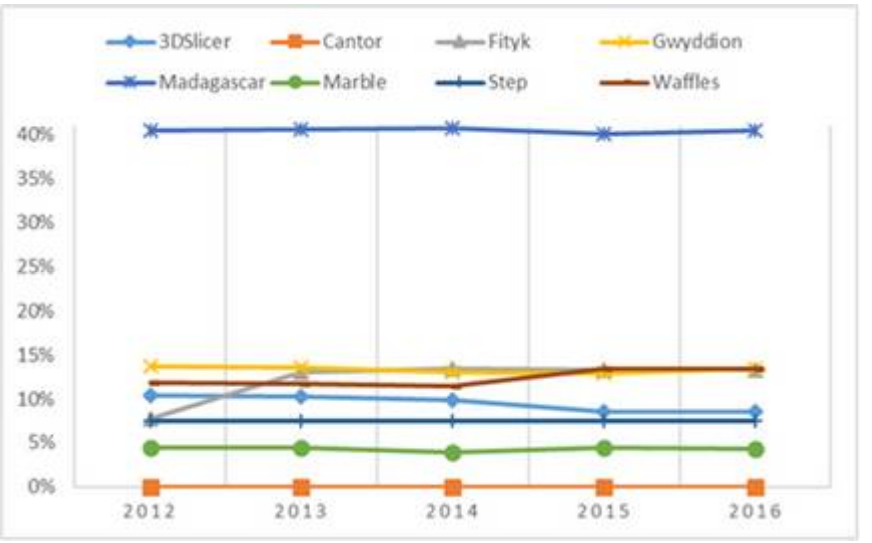


Figure 5: Percentage of For-loops with data dependency in all studied scientific systems

percentage of data dependency throughout the 5-year period, while Cantor maintains 0% data dependency for the 5-year period. Fityk had an increase in data dependency between 2012 and 2013, but then stayed flat. Waffles saw an increase over the recent years for data dependency, and 3DSlicer saw a decrease in data dependency.

Figure 6 shows the percentage of for-loops with jump statements. Marble saw the most drastic increase, while systems like Cantor and 3DSlicer saw decreases. While most of the jump statements come from the use of break, some developers are still using goto, such as the developers of Gwyddion, where goto accounts for about 2% of the jump statements in each version of the years studied despite recommendations to avoid its use.

These results indicate that scientific developers are writing their code in ways that are not improving their parallelizability. One reason for this might be due to bad programming habits or a programming background lacking in the knowledge or experience needed to write programs that can be easily parallelized.

The developers may not realize that such issues exist within their code, and one goal of this study was to help make these developers aware of potential issues that they may face when they try to adapt their code to utilize multicore architecture. Another reason might be that the developers know that there are issues with the parallelizability of their code, but choose to ignore the issues and leave them uncorrected.

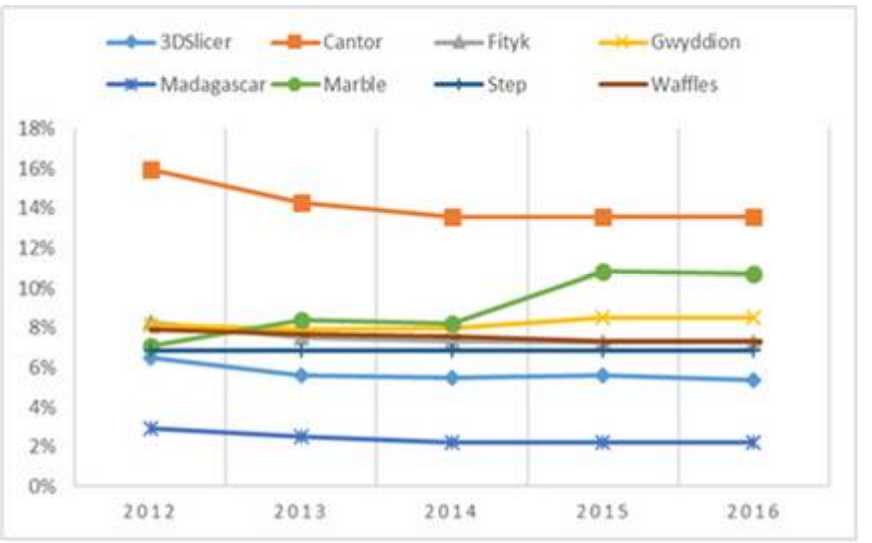


Figure 6: Percentage of For-Loops with jumps (breaks and gotos) in all studied scientific systems

8 Conclusion

This study empirically examined the challenges scientific software systems have to face before being able to take full advantage of multicore technology. Twelve open source scientific systems were studied, comprising over 5.4 million lines of code and containing more than 84.5 thousand for-loop statements. There are no other studies of this type that have been conducted on scientific software systems in the context of source-to-source parallelization currently in the literature.

We found that the greatest inhibitor to parallelizing scientific software systems is the presence of function calls with side effects, followed closely by data dependency. As such, more attention needs to be placed on dealing with function-call and data dependency inhibitors if a large amount of parallelization is to occur in scientific software systems so they can take better advantage of modern multicore hardware.

Our results show some indication that this is a different trend from the trends general-purpose software systems show. That is, function calls with side effects were larger than the data dependency we detected in this study.

Additionally, we empirically showed that coding style can play a big role in advancing a system's parallelizability, with developers of scientific systems causing more challenges to the parallelization process by using patterns that cause function side-effects. That is at least partially due to development teams not focusing on developing software in a way that could one day take advantage of parallel architectures. However, the recent ubiquity of multicore processors gives rise to the need to educate scientific software developers and make them aware of inhibitors preventing the parallelization of their code.

In order to see how scientific systems are adapted over time to take advantage of multicore architecture, we extended the work by empirically examining the source code over a 5-year period of history for 8 of the original 12 open source scientific software systems to find the number of inhibitors and free for-loops in order to discover whether or not the systems are becoming more parallelizable over time, or if they are facing more challenges towards the adaptation to utilize multicore architecture as they grow and evolve over time.

Our results showed that open source scientific systems are not, in general, becoming more parallelizable as they evolve over time. Instead, the average percentage of free for-loops found in the systems saw a decrease of about 1% between the years 2012 and 2016 meaning that these scientific systems are becoming slightly less parallelizable as they grow instead of more parallelizable. When the systems are examined individually, many show a flat trend where their parallelizability is neither increasing nor decreasing by a significant amount.

From the results of this work, we recommend that scientific software research communities develop techniques that assist in removing jumping statements along with the identification of functions with side effects (in the context of parallelization).

Acknowledgements

This work was supported in part by a grant from The University of Wisconsin-Colleges and UW-Fox Valley.

References

[1] S. M. Alnaeli, A. A. Taha, and T. Timm, "On the Prevalence of Function Side Effects in General Purpose Open Source Software Systems," *Computer and Information Science*, R. Lee, Ed., ed Cham: Springer International Publishing, pp. 149-166, 2016.

[2] S. M. Alnaeli, J. I. Maletic, and M. L. Collard, "An Empirical Examination of the Prevalence of Inhibitors to the Parallelizability of Open Source Software Systems," *Empirical Software Engineering,* 21:1272-1301, 2016.

[3] U. K. Banerjee, *Dependence Analysis for Supercomputing*, Kluwer Academic Publishers*,* 1988.

[4] B. Barney, *Introduction to Parallel Computing.* Available: https://computing.llnl.gov/tutorials/parallel_comp/#Models, 2012.

[5] A. J. C. Bik and D. Gannon, "Automatically Exploiting Implicit Parallelism in Java," *Concurrency - Practice and Experience,* pp. 579-619, 1997.

[6] M. L. Collard, M. J. Decker, and J. I. Maletic, "Lightweight Transformation and Fact Extraction with the srcML Toolkit," Presented at the Proceedings of the 2011 IEEE 11th International Working Conference on Source Code Analysis and Manipulation, 2011.

[7] L. Feng, Automatic Parallelization in Graphite, Available: http://gcc.gnu.org/wiki/Graphite/Parallelization, 2009.

[8] C. Ghezzi and M. Jazayeri, *Programming Language Concepts*, Wiley, 1982.

[9] J. E. Hannay, C. MacLeod, J. Singer, H. P. Langtangen, D. Pfahl, and G. Wilson, "How do Scientists Develop and Use Scientific Software?," *ICSE Workshop on Software Engineering for Computational Science and Engineering, 2009. SECSE '09*, pp. 1-8, 2009.

[10] D. A. P. J. L. Hennessy *Computer Architecture: A Quantitative Approach*, Morgan Kaufman Publishers, San Francisco*,* 2006.

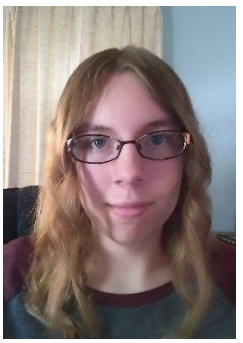
[11] Intel., "Automatic Parallelization with Intel Compilers," Available: http://software.intel.com/en-us/articles/automatic-parallelization-with-intel-compilers/, 2010.

[12] N. Nadgir, "Using OpenMP to Parallelize a Program," Available: http://developers.sun.com/solaris/articles/openmp.html, 2001.

[13] D. S. Nikolopoulos, C. D. Polychronopoulos, E. Ayguad´e, J. U. Labarta, and T. S. Papatheodorou, "The Trade-off between Implicit and Explicit Data Distribution in Shared-Memory Programming Paradigms," presented at the ICS '01, Sorrento, Italy, 2001.

[14] Oracle, "Inhibitors to Explicit Parallelization," Available: https://docs.oracle.com/cd/E19205-01/8195262/aeuji/index.html, 2010.

Saleh M. Alnaeli is an Assistant Professor in the Department of Computer Science at The University of Wisconsin-Fox Valley. His research interests focus on software evolution, automatic parallelization, software security, and mining software repositories. Dr. Alnaeli obtained his PhD in Computer Science from The Kent State University. He also received the MS in Computer Science from Technical University of Ostrava, Czech Republic 2006, and the BS in Computer Science from The University of Zawia, Libya 1999.



Melissa M. Sarnowski is an undergraduate student in the Department of Computer Science at University of Wisconsin-Fox Valley. Her research interests are centered around software evolution and security. She has authored multiple referred publications in the areas of analysis, security, and parallelizability of software. She is leading the computer club at UW-Fox Valley and a member of the software engineering lab lead by Professor Alnaeli at UW-Colleges.

Calvin Meier is an undergraduate student in the Department of Computer Science at University of Wisconsin-Oshkosh. His area of interest for research is software analysis. He was a member of the computer club and of the software engineering lab lead by Professor Alnaeli prior to his transfer to the UW-Oshkosh campus.

Mark Hall is an Associate Professor in the Department of Computer Science at University of Wisconsin-Marathon County. He coordinates the UW Colleges' programming teams for competitions and his area of interest for research is Computer Science education.

Instructions for Authors

The International Journal of Computers and Their Applications is published multiple times a year with the purpose of providing a forum for state-of-the-art developments and research in the theory and design of computers, as well as current innovative activities in the applications of computers. In contrast to other journals, this journal focuses on emerging computer technologies with emphasis on the applicability to real world problems. Current areas of particular interest include, but are not limited to: architecture, networks, intelligent systems, parallel and distributed computing, software and information engineering, and computer applications (e.g., engineering, medicine, business, education, etc.). All papers are subject to peer review before selection.

A. Procedure for Submission of a Technical Paper for Consideration

1. Email your manuscript to the Editor-in-Chief, Dr. Fred Harris, Jr., Fred.Harris@cse.unr.edu.

2. Illustrations should be high quality (originals unnecessary).

3. Enclose a separate page (or include in the email message) the preferred author and address for correspondence. Also, please include email, telephone, and fax information should further contact be needed.

B. Manuscript Style:

1. The text should be **double-spaced** (12 point or larger), **single column** and **single-sided** on 8.5 X 11 inch pages.
2. An informative abstract of 100-250 words should be provided.
3. At least 5 keywords following the abstract describing the paper topics.
4. References (alphabetized by first author) should appear at the end of the paper, as follows: author(s), first initials followed by last name, title in quotation marks, periodical, volume, inclusive page numbers, month and year.
5. Figures should be captioned and referenced.

C. Submission of Accepted Manuscripts

1. The final complete paper (with abstract, figures, tables, and keywords) satisfying Section B above in **MS Word format** should be submitted to the Editor-in-Chief.
2. The submission may be on a CD/DVD or as an email attachment(s) . **The following electronic files should be included:**
    - Paper text (required).
    - Bios (required for each author). Integrate at the end of the paper.
    - Author Photos (jpeg files are required by the printer, these also can be integrated into your paper).
    - Figures, Tables, Illustrations. These may be integrated into the paper text file or provided separately (jpeg, MS Word, PowerPoint, eps).

3. Specify on the CD/DVD label or in the email the word processor and version used, along with the title of the paper.

4. Authors are asked to sign an ISCA copyright form (http://www.isca-hq.org/j-copyright.htm), indicating that they are transferring the copyright to ISCA or declaring the work to be government-sponsored work in the public domain. Also, letters of permission for inclusion of non-original materials are required.

Publication Charges

After a manuscript has been accepted for publication, the contact author will be invoiced for publication charges of **$50.00 USD** per page (in the final IJCA two-column format) to cover part of the cost of publication. For ISCA members, $100 of publication charges will be waived if requested.

January 2014