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 23, No. 3, Sept. 2016 **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

MINING DECISION TREES AS TEST ORACLES FOR JAVA BYTECODE

Weifeng Xu
Bowie State University, Bowie, MD USA

Tao Ding
University of Maryland, Baltimore County, Baltimore, MD USA

Dianxiang Xu
Bowie State University, Bosie, ID USA

Omar El Ariss
The Pennsylvania State University-Harrisburg, Middletown, PA USA

Abstract

Code-based test generation can automatically produce a large volume of test inputs. However, it is difficult to determine the test oracle for each of the test inputs. This paper presents a mining approach to building a decision tree model according to the test inputs generated from Java bytecode so that the model can be used as a source of test oracles for new test inputs. This approach converts Java bytecode into the Jimple representation, extracts predicates from the control flow graph of the Jimple code, and uses these predicates for generating test inputs and as attributes for organizing training data to build a decision tree. Our case studies show that the mining approach generated accurate behavioral models and that test oracles derived from these models can kill 94.67% of the mutants with injected faults.

Key Words: Software testing, test oracle, mining, decision tree, Jimple.

1 Introduction

As software testing is a labor-intensive activity, significant research has been directed to test automation. One of the approaches is code-based test generation, which automatically generates test inputs from source code or compiled code. For example, search–based test generation [1-2, 28-29] utilizes heuristic search algorithms, such as hill climbing and genetic programming, to search for the best-fit data as test inputs by evaluating fitness function in terms of the source code structure of the Unit Under Test (UUT). An advantage of such code-based test generation is that it can automatically generate a large number of test inputs for exercising the UUT. However, it is often difficult to determine the expected result for each generated test input to determine whether its execution passes or fails. This is known as the test oracle issue. A test oracle [32] verifies whether the actual result of a test case matches its expected result. For example, the assertion `assertEquals (“Equilateral”, new Trianlge (7,7,7).getTriType())` in JUnit checks that a triangle program reports the correct triangle type `Equilateral` for the given length of 7 for all three sides.

To address the test oracle issue of auto-generated test inputs, the paper proposes a new data mining approach to building a heuristic behavioral model (in the form of decision tree [38]) that represents the estimated expected results of test inputs. An estimated expected result for a given test input will be retrieved from the generated model. The execution of a test case is considered as a “pass” if the actual result matches the estimated result in the model. Our approach uses search-based techniques for generating a large number of test inputs from the Java bytecode of UUT. Therefore, it does not depend on the availability of source code. From the generated test inputs, we chose a small subset of the test inputs (e.g., 10%) as training data for building the behavioral model. For the remaining test inputs, we derive their expected results from the model so that test oracles can be automated. As shown in Figure 1, our

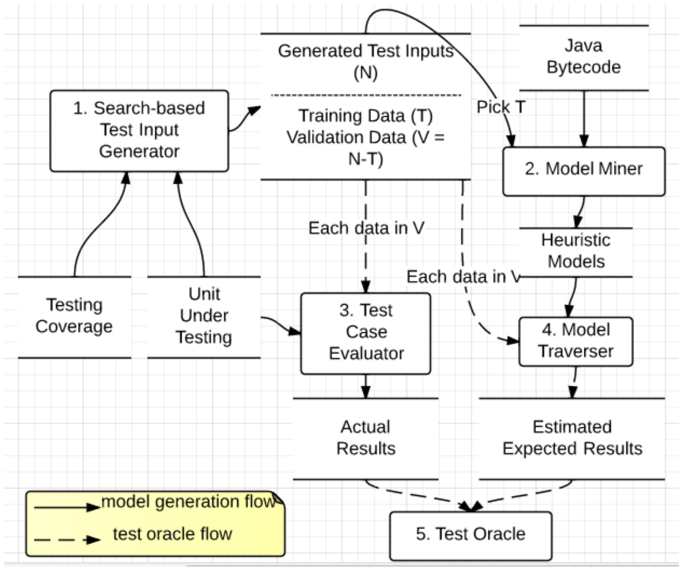


Figure 1: Overview of our approach

approach consists of the following main components:

1) Search-based Test Input Generator. This generates N test inputs from the given UUT's Java bytecode according to a test coverage criterion. Each input X is a vector composed of the input variables <x_1, x_2, ..., x_n> of the UUT. The test input generator uses Jimple as an intermediate representation of Java bytecode.
2) Model Miner. This extracts necessary information (e.g., predicates) from the Java bytecode to generate a decision tree model M from the training data. The initial training data T is a subset of test inputs N.
3) Test Case Evaluator. This executes the UUT to produce actual results for the remaining test inputs `V = N - T`. Let `UUT(X)` denote the actual result of test input `X`.
4) Model Traverser. This searches the decision tree for estimated expected result of test input X. Let MT(X) denote the estimated result of test input X.
5) Test Oracle. For each test input in V, it compares the estimated expected with the actual result using assertion `assertEquals(MT(X), UUT(X))`.

Our approach is the first attempt to mine decision tree models for test inputs auto-generated from Java bytecode with respect to different coverage criteria. We used fault injection for the evaluation of the proposed approach. The result of the empirical study shows that using the mined test oracles, the generated tests were able to kill 94.67% of the mutants.

The rest of this paper is organized as follows: Section 2 introduces a running example and some basic concepts. Sections 3, 4, and 5 describe predicate analyzer, rule-based test inputs generator, and model miner, respectively. Section 6 describes the empirical studies. Section 7 reviews the related work. Section 8 concludes this paper.

2 Motivating Example

We use the classical Triangle problem [17] as a running example. Given three positive integers that represent the lengths of three sides of a triangle, the Triangle program reports the triangle type, `Equilateral (Type 1)`, `Isosceles (Type 2)`, `Scalene (Type 3)`, or `NotATriangle (Type 4)`. We use Jimple to analyze the structure of the source code for test input generation and use the same Jimple predicate structure as attributes for mining decision tree models. Jimple is a bridge between input generation and model mining.

2.1 Jimple

Java bytecode is a stack-oriented language, which pops data from the top of the stack, and pushes data back on the top of the stack. For example, the bytecode instruction `iadd` pops two integer values from Java virtual machine stack [16] and pushes their sum. Bytecode instructions have an implicit effect on the evaluation stack. For the analysis of bytecode, an effective approach is to use an intermediate representation of bytecode, such as 3-address instructions, where each instruction has explicitly named operands. For example, an addition operation would be represented as something like `x = a + b`.

Our approach uses Jimple [33], a 3-address intermediate representation that has been designed to simplify analysis and transformation of Java bytecode. Each statement of Jimple refers explicitly to the variables it uses. One of the essential characteristics is that statements are restricted to the least number of operands (2 in most cases, such as for arithmetic expressions), and these operands must be either constants or locals. For example, for a given Java statement `x = a + b + c`, the corresponding two Jimple statements are `S1: $i3 = i0 + i1` and `S2: i4 = $i3 + i2`, where `i0`, `i1`, `i2`, `$i3`, and `i4` are local variables of Jimple code and `i0`, `i1`, `i2`, and `i4` correspond to variables `a`, `b`, and `c` in the given Java statement. Variables with a "`$`" sign are intermediate variables, e.g., `$i3`. Another benefit of using Jimple is that predicates with multi-conditions of Java source code will be represented by multi-level conditions as shown in Table 1. The Soot package [20, 40] can be used to construct the control flow graph (CFG) of Jimple code.

Table 1: A multi-condition in Java represented in Jimple code

Java Source Code	Jimple Code

```
if (condition1 &&
    condition2 &&
    condition3{
      Statement 1;
      Statement 2;
      …
}
```
	```
	if (condition 1){
	  if (condition2){
	    if (condition3) {
	          Statement 1;
	          Statement 2;
	          …
	    }
	  }
	}
	```

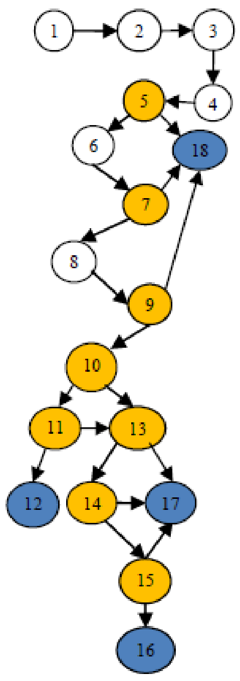
To facilitate our discussion, Table 2 presents the Java source code, Jimple code, and the CFG (control flow graph) of the Jimple code for the `Triangle` program. Each CFG node is corresponding to a Jimple statement. Note that each predicate (colored in yellow) in the source code is represented by a set of predicates in the Jimple code. For example, the Java statement [1], `(a < b + c) && (b < a + c) && (c < a + b)`, is decomposed into three 3-address intermediate representations, i.e., nodes [32] `(i0 >= $i3)`, [17] `(i1 >= $i4)`, and [33] `(i2 >= $i5)` at different levels in the CFG of Jimple code.

2.2 Auto-generated Test Inputs and Decision Trees

Table 3 shows some examples of auto-generated test inputs `(a, b, c)` for the triangle program using the search-based approach with decision coverage of the Jimple code. The decision coverage generates two test cases for each predicate, making the predicate evaluate to be true and false, respectively. This ensures complete testing of control constructs [17]. The cover times and test input IDs record how many test inputs need to be generated to cover all predicates

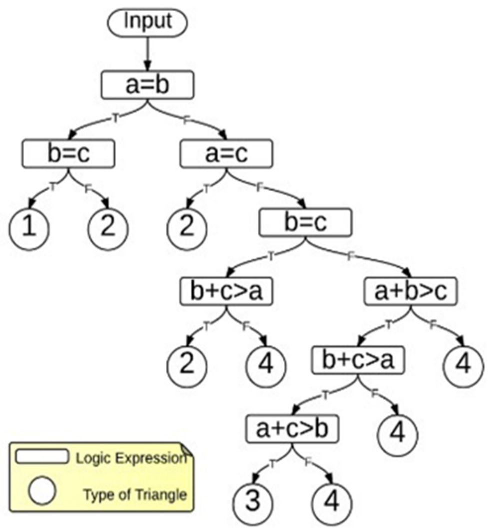
Table 2: Jimple CFG of triangle problem

CFG of Jimple	Code

	`[1]i0 := @parameter0: int`
	`[2]i1 := @parameter1: int`
	`[3]i2 := @parameter2: int`
	`[4]$i3 = i1 + i2`
	`[5]if i0 >= $i3 goto return 4`
	`[6]$i4 = i0 + i2`
	`[7]if i1 >= $i4 goto return 4`
	`[8]$i5 = i0 + i1`
	`[9]if i2 >= $i5 goto return 4`
	`[10]if i0 != i1 goto 13`
	`[11]if i1 != i2 goto 13`
	`[12]return 1`
	`[13]if i0 == i1 goto return 2`
	`[14]if i0 == i2 goto return 2`
	`[15]if i1 == i2 goto return 2`
	`[16]return 3`
	`[17]return 2`
	`[18]return 4`

	`[1]int getTriType (a,b c)`
	`[2]if((a<b+c) && (b<a+c) && (c<a+b)){`
	`[3]    if(a==b && b==c)`
	`[4]        return 1;`
	`[5]    else if (a!=b && a!=c &&b!=c)`
	`[6]        return 3;`
	`[7]    else`
	`[8]        return 2;`
	`[9]}`
	`[10]   else`
	`[11]       return 4;}`

Table 3: Examples of auto-generated test inputs for the triangle program using search-based approach with the corresponding conceptual decision tree

Cover Times	Test input ID	a	b	c	A Conceptual Decision Tree that Classifies Triangles
Once	1	7	7	7	
	2	11	7	3	
	3	8	11	19	
	4	11	7	3	
	5	22	22	9	
	6	30	43	30	
	7	22	13	13	
	8	13	16	20	
..	…				
…	…	…	…	…	
Fourth	…				
	13	33	33	45	
	14	52	30	52	
	15	31	47	47	
	16	27	28	22	

once, twice, three times, etc. For example, 8 test inputs need to be generated to cover each Jimple predicate once. Intuitively, the triangle type can be classified by conditions in Java source code. For example, an `Equilateral` is classified by two conditions: `a == b` and `b == c` (i.e., statement 3 in Java source code). These conditions correspond to `i0 == i1` and `i1 == i2` in Jimple code (i.e., nodes [41] and [15]) by observation. The last column of Table 3 shows a conceptual decision tree that represents the knowledge of classifying triangles. The intermediate nodes of the decision tree are conditions and the leaves of the tree indicate the types of triangles. Our goal is to build such a decision tree with a set of training data. The training data are designed by a set of conditions. If training data fits the classification with high accuracy, then we can reasonably assume that for any newly generated test input, an estimated expected result can be obtained by traversing the path from the root to a leaf.

The following sections address the critical issues of our approach. First, how to generate test inputs automatically based on Jimple code (Sections 3 and 4). Second, how to extract appropriate conditions from Jimple code to build a decision tree (Sections 3 and 5). Third, how to determine the accuracy of decision trees, and more important, how effective are the test oracles derived by the mined decision tree in detecting faults in the given program (Section 6).

3 Predicate Analyzer

Our approach uses a predicate analyzer to extract and analyze conditions from Jimple code for both generating test inputs and building decision trees. Specifically, predicate analyzer examines the Jimple statements to discover relationships between Jimple input variables and variables in predicates using variable dependency analysis. These relationships are crucial as (1) to generate test inputs for a given path, a set of rules we have defined are the guidelines for changing values in predicates for traversing the path. As Jimple variables in predicates are often intermediate variables, they need to be backtracked to Jimple inputs by utilizing these relationships (addressed in section 4). (2) For mining test inputs to build the decision tree, each generated input is represented as a set of attributes. These attributes are Jimple predicates, and thus, these relationships are needed again for transferring each generated input to attributes (Section 5). In this section, we first formally define the path used in the paper and then discuss variable dependencies.

3.1 Tagged Path

A path derived from a CFG contains a sequence of statements. For example, `p1`: [1]→[2]→[3]→[4]→[5]→[6]→[7]→[8]→[9]→[10]→[11]→[12] is one of the paths derived from Jimple CFG for testing an equilateral triangle. To execute all statements in `p1`, we need to adjust values of all variables involved in the predicates so that the predicates can produce desired values for reaching the last statement. Thus, we need to achieve the desired outcome for each predicate in `p1` as follows:

`p'1`: $[1]\rightarrow[2]\rightarrow[3]\rightarrow[4]\rightarrow[5]\xrightarrow{f}[6]\rightarrow[7]\xrightarrow{f}[8]\rightarrow[9]\xrightarrow{f}[10]\xrightarrow{f}[11]\xrightarrow{f}[12]$,

where f (false) is the expected outcome of the corresponding predicate. Such a path is called a tagged path.

Definition 1 (Tagged Edge). A tagged edge is defined as $v\xrightarrow{f}u$, where v is the source of the edge that represents a predicate in a statement, o is a tagged value for v, which represents the desired outcome of v (i.e., true or false), and u is the reachable statement if the assertion (v==o) returns true.

Definition 2 (Tagged Path). A tagged path is a sequence of edges that has at least one tagged edge.

Table 4 shows tagged paths of the `Triangle` program based on decision coverage. In the tagged path `p12`, the tagged edge $[5]\xrightarrow{f}[18]$ indicates that node 5 is a predicate and its outcome `(i0 >= $i3)` must be true in order to reach node 18.

3.2 Variable Dependency Tree

Consider a simply tagged path `p12`: $[1]\rightarrow[2]\rightarrow[3]\rightarrow[4]\rightarrow[5]\xrightarrow{f}[18]$ in Table 4, the goal is to find a test input to exercise this path (i.e., find a triangle type of "NotATriangle"). As $[5]\xrightarrow{f}[18]$ is the only tagged edge, the path will be covered if a test input makes the constrain in [32]: `i0 >= $i3` as `true`. To generate a test input for covering `p12`, however, we need to determine which input variables are associated with `$i3` so that we can adjust these input variables to meet the constraint. It is not difficult to see that `$i3` is associated with input variables `i1` and `i2` by backtracking `$i3`.

Variable dependency analysis is the process of backtracking input variables for given intermediate Jimple variables for make `assertion (v == o)` true. The dependency analysis starts with identifying write-dependency and read-dependency relations. Given an assignment statement that involves variables i_m and i_n, i_m is write-dependent on i_n if and only if i_m writes a resource that i_n reads. For two assignment statements `S1` and `S2` that use variable i_n, and `S1` precedes `S2` in execution, the variable i_n in `S2` is read-dependent on the i_n in `S1` if and only if i_n in `S2` read a resource that i_n writes in `S1`. Table 5 shows three examples. The third example consists of two statements with one read- and one write-dependency relations.

Variable dependency analysis is also utilized for mapping Jimple conditions to a recognizable attribute set. For example, a Jimple condition `i0 > = $i3` cannot be directly used as an attribute for mining decision tree models as `i0` and `$i3` are Jimple variables, variable dependency analysis provides mapping information converting the Jimple condition to the

Table 4: Tagged paths for triangle problem

Goal	ID	Path
Equilateral	1	[1]→[2]→[3]→[4]→[5] $\xrightarrow{f}$[6]→[7]$\xrightarrow{f}$[8]→[9]$\xrightarrow{f}$[10] $\xrightarrow{f}$[11]$\xrightarrow{f}$[12]
Isosceles	2	[1]→[2]→[3]→[4]→[5] $\xrightarrow{f}$[6]→[7]$\xrightarrow{f}$[8]→[9]$\xrightarrow{f}$[10] $\xrightarrow{f}$[13] $\xrightarrow{f}$[14]$\xrightarrow{f}$[15]$\xrightarrow{f}$[17]
	3	[1]→[2]→[3]→[4]→[5] $\xrightarrow{f}$[6]→[7]$\xrightarrow{f}$[8]→[9]$\xrightarrow{f}$[10] $\xrightarrow{f}$[13] $\xrightarrow{f}$[14]$\xrightarrow{f}$[17]
	4	[1]→[2]→[3]→[4]→[5] $\xrightarrow{f}$[6]→[7]$\xrightarrow{f}$[8]→[9]$\xrightarrow{f}$[10] $\xrightarrow{f}$[11] $\xrightarrow{f}$[13]$\xrightarrow{f}$[17]
	5	[1]→[2]→[3]→[4]→[5] $\xrightarrow{f}$[6]→[7]$\xrightarrow{f}$[8]→[9]$\xrightarrow{f}$[10] $\xrightarrow{f}$[13] $\xrightarrow{f}$[17]
	6	[1]→[2]→[3]→[4]→[5] $\xrightarrow{f}$[6]→[7]$\xrightarrow{f}$[8]→[9]$\xrightarrow{f}$[10] $\xrightarrow{f}$[11] $\xrightarrow{f}$[13] $\xrightarrow{f}$[14]$\xrightarrow{f}$[15]$\xrightarrow{f}$[17]
	7	[1]→[2]→[3]→[4]→[5] $\xrightarrow{f}$[6]→[7]$\xrightarrow{f}$[8]→[9]$\xrightarrow{f}$[10] $\xrightarrow{f}$[11]$\xrightarrow{f}$[13]$\xrightarrow{f}$[14]$\xrightarrow{f}$[17]
Scalene	8	[1]→[2]→[3]→[4]→[5] $\xrightarrow{f}$[6]→[7]$\xrightarrow{f}$[8]→[9]$\xrightarrow{f}$[10] $\xrightarrow{f}$[13]$\xrightarrow{f}$[14]$\xrightarrow{f}$[15]$\xrightarrow{f}$[16]
	9	[1]→[2]→[3]→[4]→[5] $\xrightarrow{f}$[6]→[7]$\xrightarrow{f}$[8]→[9]$\xrightarrow{f}$[10] $\xrightarrow{f}$[11]$\xrightarrow{f}$[13]$\xrightarrow{f}$[14]$\xrightarrow{f}$[15]$\xrightarrow{f}$[16]
NotATriangle	10	[1]→[2]→[3]→[4]→[5] $\xrightarrow{f}$[6]→[7]$\xrightarrow{f}$[8]→[9] $\xrightarrow{f}$[18]
	11	[1]→[2]→[3]→[4]→[5] $\xrightarrow{f}$[6]→[7] $\xrightarrow{f}$[18]
	12	[1]→[2]→[3]→[4]→[5] $\xrightarrow{f}$[18]

Table 5: Examples of dependency relations

ID	Variable	Statement	Dependency Relation
1	i1, i2	S1:i2 = i1 + C, (C: constant)	i2 is write-dependent on i1
2	i1, i2 , i3	S1:i3 = i2 + i1	i3 is write-dependent on i1 and i2
3	i3	S1: i3 = i0 + i1 S2: i4 = i2 + i3	i3 in S2 is read-dependent on i3 in S1 i3 is write-dependent on i1, etc.

recognizable attribute, i.e., `a < b + c` so that each generate input can be transferred into training data.

Variable dependency can be graphically captured in a variable dependency tree (VDT). It is a tree structure used for tracking read- and write-dependency relations among all variables in the Jimple code. A VDT consists of a set of tree units, where each tree unit corresponds to an assignment statement of Jimple code. Each tree unit consists of three nodes, one parent node and two child nodes, and an operator between the two children. Edges in tree units from the parent nodes to child nodes are write-dependency relations. Relations between tree units are read-dependency relations. Figure 2 shows a VDT that consists of two tree units (contained in two squares) constructed from the Java statement `x = a + b + c`. These two tree units of the VDT represent the two corresponding Jimple statements `S1: i3 = i0 + i1` and `S2: i4 = i3 + i2` shown in the third example in Figure 2. Arrows in the VDT represent four write-dependency relations and the dashed line between `S1` and `S2` represents a read-dependency relation.

The algorithm in Table 6 describes the procedure for building VDTs from the Jimple code of a UUT. A tree unit is denoted as `t(root, leftChildNode, rightChildNode, operator)` where each node contains a Jimple variable. The algorithm recursively expands the child nodes with tree units until all leaves of the tree are test input variables in the Jimple code. Figure 3 shows the VDTs of the triangle Jimple code. The leaves in the VDTs are `i0, i1`, and `i2`, which are the input variables.

4 A Rule-Based Test Inputs Generator

Our rule-based search algorithm (RBA) utilizes a set of pre-path. The approach starts with a randomly generated test input for a given path derived from the UUT. The path consists of a sequence of statements in the form of Jimple code. The path and the initialized input value (as a seed) are placed in a controlled testing environment where Jimple code can be executed. By checking predicate values along the running path, relevant rules are applied as guidelines for adjusting the seed value as the new input of next iteration. The process stops when all the statements in the path are executed. In this case, a test input is generated successfully. As shown in Figure 4, our approach consists of the following main components:

- Jimple Executor: Execute the intermediate representation of Java bytecode in the form of Jimple statements. A

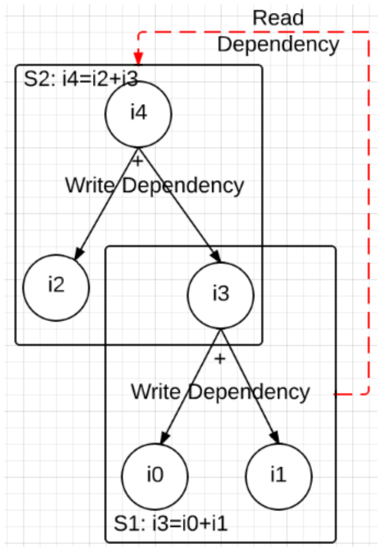


Figure 2: Variable dependency analysis

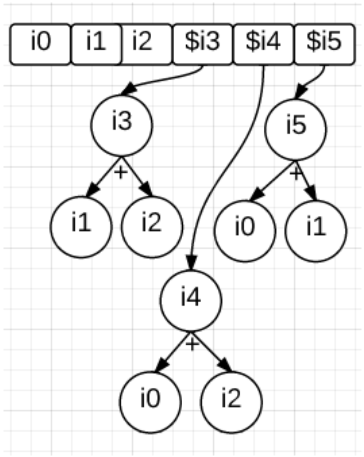


Figure 3: VDTs of Path p1

Table 6: Algorithm of building VDTs

Algorithm : Building VDTs	
Inputs : *J* : Jimple Code of a given path	
Outputs : *L*: A list of variables point to the root of VDTs contained in the variable	
1	*procedure* buildVDTs(*J*)
2	*for each* assignment statement *S* of *J*

3	*t(v, l, r, o)* ⟵create a tree unit with the root contains *v*
	// v is the root, l and r are the left and right of root,
	//o is the operator of the root

4	*if* *l* or *r* not input parameters
5	goto step 3 with *l* or *r* as a new root
6	end if
7	add *v* to *L* and point v to the tree
8	end for
9	end procedure

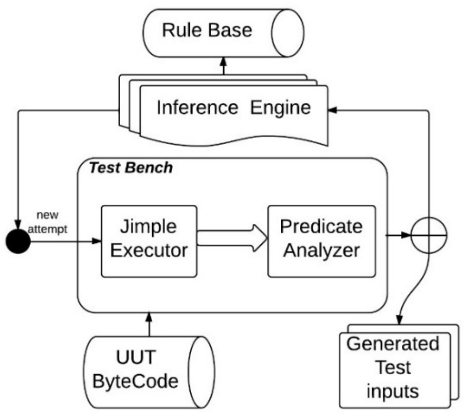


Figure 4: Overview of the rule-based test inputs generator

sequence of Jimple statements is a path derived from CFG of a UUT.

- Predicate Analyzer: Analyze variable dependency of Jimple statements and monitors the outcomes of predicates at runtime.
- Rule Base: A set of predefined rules for test input generation. To make `(a > b)` as *true*, for example, rules can either increase the value of `a` or decrease value of `b`.
- Inference engine: It selects rules from the rule base for modifying current seed value x (e.g., increasing or decreasing x) to generate a better seed value x' = x + Δx and achieve the expected outcomes of predicates.

4.1 Rules as Search Guidelines

We use a set of pre-defined search rules that are based on predicates and their expected evaluation outcomes. These rules are presented in Table 7. For a given predicate in the CFG of Jimple code, there are multiple evaluation results. For each outcome, there are several rules for instructing a predicate to produce the expected result. For example, for the first

Table 7: The pre-defined heuristic search rules for predicates

ID	Predicate	Expected Evaluation Outcomes	Advising Rules
	`i0 > i1`	`(i0 > i1) = true`	`(i0↑, i1)  (i0, i1↓)`
		`(i0 > i1) = false`	`(i0↓,b) (i0, i1↑)`
	`i0 == i1`	`(i0 == i1)= true`	`(i0↓D, i1)( i0, i1↑D)`

		`(i0 == i1)= false`	`(i0↑,i1) (i0,i1↑)`
			`(i0↓,i1) (i0, i1↓)`

	`i2 = i0 + i1`	`i2 ↑`	`(i0↑, i1) (i0, i1↑)`
		`i2 ↓`	`(i0↓, i1) (i0, i1↓)`
	`i2 = i0 - i1`	`i2 ↑`	`(i0↑, i1) (i0, i1↓)`
		`i2 ↓`	`(i0↓, i1) (i0, i1↑)`
	`i2 = i0 * i1`	`i2 ↑`	`(i0↑, i1) (i0, i1↑)`
	`(i0>0, i1>0)`	`i2 ↓`	`(i0↓, i1) (i0, i1↓)`
	`i2 = i0 / i1`	`i2 ↑`	`(i0↑, i1) (i0, i1↓)`
	`(i0>0, i0 > 0)`	`i2 ↓`	`(i0↓, i1) (i0, i1↑)`
	`..`	`..`	`..`
	`s0>s1`	`(s0 >s1) = true`	`(s0[k]↑, s1)  (s0, s1[l] ↓)`
		`(s0 > s1) = false`	`(s0[k]↓,s2)  (s0, s1[l]↑)`

* *i0, i1*, *i2*: integers; *s0, s1*: strings; D: a constant value; ↑*: value increasing;* ↓*: value decreasing*

predicate `i0 > i1` in Table 7, there are two possible evaluation results, `true` and `false`. To produce the outcome of `true`, the rules indicate that we can either increase the value of `i0` and keep the value of *i1* unchanged or increase the value of `i1` and keep the value of `i0` unchanged, i.e., `(i0↑, i1)` or `(i0, i1↑)`.

A predicate tree (PT) is used to represent predicates of input variables. The root of a PT is a predicate statement. The direct children of the root are variables in the predicates and the operator is the relation between the two children. Non-input variables will be expended using VDTs. Thus, the leaves of the PT must be Jimple input variables, and the inner nodes are intermediate Jimple variables. In Figure 5, the tree on the left-hand side shows the PT built from the aforementioned path `P`. The root of the PT is [32]. Its children are variables `i0` and `$i3`. The intermediate variable is represented by two new nodes `i1`, `i2`, and their operator, i.e., `$i3 = i1 + i2`. To instruct the execution of a UUT following a given path *P*, the test input generator uses rules in Table 7. The tree on the right-hand side of Figure 5 shows the result of applying rule 1 `(i0↑, i3)` *or* `(i0, i3↓)` and rule 5 `(i1↓, i2) (i1, i2↓)`. For the predicate statement `i0 >= $i3` and the expected outcome of *true*, we either increase the value of `i0` or decrease the value of `$i3`. For the statement `$i3 = i1+i2` and the expected outcome of decreasing value of `$i3`, either `i1` or `i2` needs to be decreased. Here, we use up- and down- arrows (e.g., `i0↑, 11↓, i2↓`*)* to represent the directions of value changing.

To handle the string type, we treat a string as an array of characters, which are represented as integers (i.e., their Unicode). For example, to generate a string `s = "abc"`, we simply assign each character to *s*, i.e., `s[0] = 97`, `s[1] = 98`, `s[2] = 99`, where `97, 98`, and `99` are the ASCII code of *"*`a`*"*, *"*`b`*"*, and *"*`c`*"*, respectively. Jimple facilitates the conversion by introducing the newarray function, which creates a character or array to represent a string, e.g., `s = newarray[3]`. For string comparison, Jimple returns

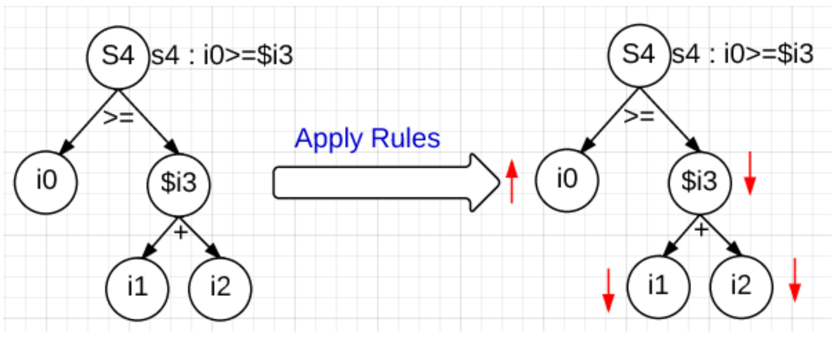


Figure 5: Apply rules to a predicate tree for generating test inputs for a designated path

predicate involving two strings with an expected outcome, i.e., `(s0 > s1) == true`. The rule indicates that to assert string `s0` is larger than `s1`, we can either increase the `kth` character of the `s0` or decrease the `lth` character of `s1`.

Our current approach does not handle loops directly. Instead, it provides an alternative way of simplifying the problem because (1) different types of loop in Java source code, including for, while, and do-while, are represented in the one simple form of `goto` instruction with a new explicit predicate. Table 8 shows an example of a simple loop in Java source code and Jimple code. The statement [16] contains an explicit predicate, i.e., `i1 < i0` and a `goto` statement. (2) loops can be transferred in VDT, which offers a fundamental way of analyzing loops for data generation.

4.2 Rule-Based Search

To generate specific inputs for traversing path □, a set of random numbers are first generated to represent the lengths of three sides, e.g., `<10, 7, 4>`. Obviously, executing the UUT with this test input does not necessarily follow □ as the test input may fail to produce the expected outcome of the predicate `(i0 >= $i3) = true`□where `$i3 = i1 + i2`. By applying rules 1 and 5 `(i0↑, 11 ↓, i2↓)`, we can generate new inputs (e.g., `<11, 6, 3>`) that may cover □. The process can be repeated as needed. Table 9 shows the algorithm for rule-based generation of test inputs to cover a given path.

Figure 6 demonstrates the process of producing test inputs with the target value of `Equilateral` (i.e., `p1` in Table 4) using RBA. The figure is divided into two parts by a dashed line. The left-hand side shows the path `p1` that contains five tagged values (all "`false`"). On the right-hand side is a tree-like structure with a backtracking edge links back to the first statement. The values in each tree node are temporary values stored in memory. For example, the line `i0 = 10`□`i1 = 20`□and `i2 = 40` in the first tree node represents three input values that are randomly generated for a given range, such as between `1` and `100`. These values are used as seeds for generating next optimized input□ These values saved in memory can be accessed by Jimple executor. After Jimple executor fetching a Jimple statement, e.g., a predicate statement [32] `i0 >= i3`□it will look for values of `i0` and `i3` to evaluate the predicate. If no variables are in the memory, Jimple executor will retrieve from VDT shown in Figure 3, i.e., `i3 = i1 +i2=20+40=60`. Thus, the Jimple executor returns `false` when evaluating `i0>=i3`. As it matches the expected outcome of predicate [32] is `false` as shown in the `p1`, the executor will fetch the next statement for execution. Otherwise, we will modify values in tree nodes by applying rules to force these values to match expected outcomes of the predicate. For example, to force predicate [33] to produce expected false values, the rule engine needs to apply rule 3 and rule 5. Thus, the tree node has three branches, i.e., either `i2` is decreased from 40 to 25, `i0` is increased

Table 8: Loop in Jimple

Java Source Code	Jimple Code
`foo(int n){`	`[1]i0 := @parameter0: int`
`  for(i=0;i<n;i++){`	`[2]i1 = 0`
`    j=10*i;`	`[3]goto statement 8`
`    if(j>50){`	`[4]i2 = 10 * i1`
`     return 1 //target`	`[5]if i2 <= 50 goto statement 7`
`    }`	`[6]return 1        //target`
`  }`	`[7]i1 = i1 + 1`
`  return 0;`	`[8]if i1 < i0 goto statement 4`
`}`	`[9]return 0`

Table 9: Algorithm for rule-based generation of test inputs

```
Algorithm: Rule-based approach for generating test inputs
Inputs: R : a set of rules
        P:  a designed path
        J : Jimple Code of a UUT
Outputs: a generated test input covers P
1  procedure testInputGenerator(R, P, J)
2   (i1, i2,.., in) ← randomly generate an input
3     while (i1, i2,.., in) != covers P
4       pt ← building PT from P
5       (i1 ↑?, i2 ↓?,.., in↓?)← Apply rules to pt
6       (i1, i2,.., in) ← adjust (i1, i2,.., in)
7     end while
8     return (i1, i2,.., in)
9  end procedure
```

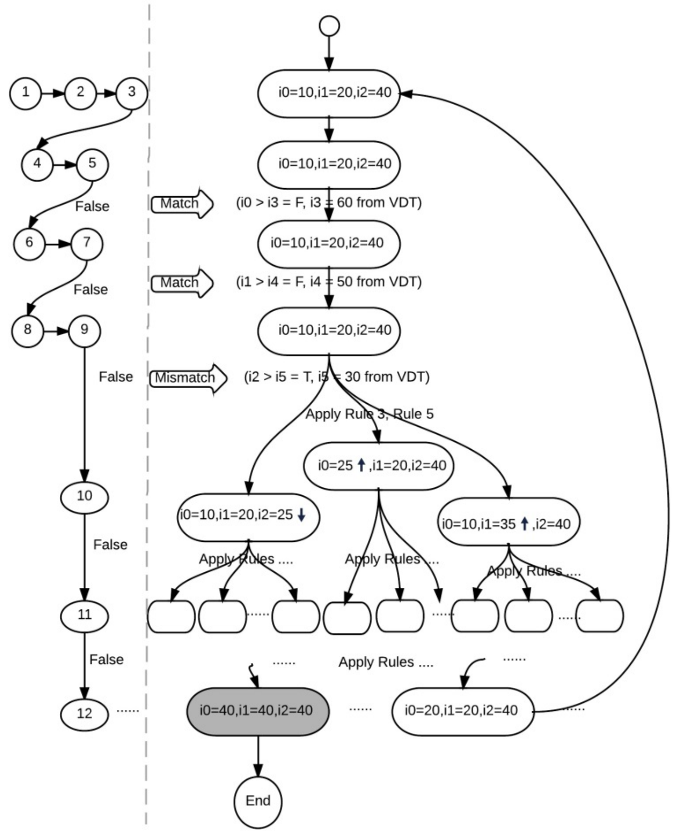


Figure 6: The process of rule-based test input generation

from `10` to `25`, or `i1` increases from `20` to `35`. Once it reaches the final statement, a tree has been built and the leaves of the tree represent generated test input data. If the last statement is reached, the data is correctly generated (shown in grey); otherwise, these data can be used as input to repeat the process.

Note that there are possible conflicting rules, i.e., one rule states that we should increase the value of variable *v*, while another states that we should decrease the value of variable `v`. The conflict rules may lead to infeasible paths. Take the two simple predicates in a path, i.e., `i0 > i1` and `i0 < i1`, with both expected true outcomes as an example: to assert `i0 > i1` true, the rule 1 indicates `(i0↑, I1↓)` and to assert `i0 < i1` true (equivalent to assert `i0 > i1` false), the rule 2 indicates (`i0↓`,`I1↑`*)*. After applying rule 1 and rule 2 to the predicates, each variable receives contradictive change requests. The contradictive change requests may lead to infeasible paths. In our empirical study, we simply exclude infeasible paths before applying the proposed input generation algorithm.

5 Model Miner

The model miner (as shown in Figure 7) classifies a set of training data to build a decision tree model. It first converts the Java bytecode of a UUT into Jimple code and extracts the predicates in Jimple code as attributes by substituting the variables of Jimple predicates for the input variables in the UUT. Then it organizes training data according to the identified attributes and builds the decision tree from the training set

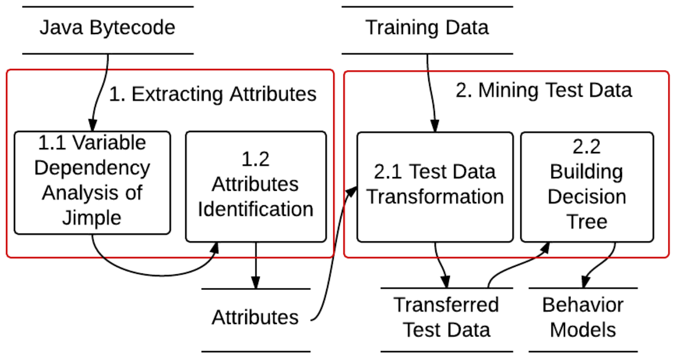

Java Bytecode
Training Data
1. Extracting Attributes
2. Mining Test Data
1.1 Variable Dependency Analysis of Jimple
1.2 Attributes Identification
2.1 Test Data Transformation
2.2 Building Decision Tree
Attributes
Transferred Test Data
Behavior Models

Figure 7: Overview of the model miner

5.1 Attributes Identification

Attributes used for designing training data are identified by replacing variables of Jimple predicates with test input variables using VDTs and replacing test input variables in the Jimple code with the corresponding input variables in Java. `i0`, `i1`, and `i2` are three input variables corresponding to input variables of Java method `getTriType(a, b, c)`, respectively. For a given *Triangle* Jimple predicate node 5: `i0 > = $i3` (refer to Table 10) the corresponding attribute `a >= b + c` is computed by replacing `$i3` with `i1+i2` based on VDTs in Figure 3 and replacing `i0`, `i1`, and `i2` with `a`, `b`, and `c`, respectively. Table 10 shows all attributes and Jimple predicates of the triangle from where these attributes were converted.

5.2 Mining Test Data

A decision tree model *M* used for test oracle is to find `Y` for a given input `X`, i.e., $X \xrightarrow{f} Y$. The root of the tree is `X` and `Y` is a leaf of the tree. Each intermediate node corresponds to one of the attributes shown in Table 10. Edges to children of the *M* are the split conditions of its parent. For example, one of the intermediate nodes is `(b + c > a)`, its left and right edges (branches) represent the conditions of classifications `(b + c > a)` and `(b + c <= a)`, respectively. We chose decision trees to describe the behavior model of a UUT due to decision tree [32] is an effective way to represent classified information based on a popular C4.5 mining algorithm [38]. The key idea of the algorithm is to calculate the highest normalized information gain of attributes and then build a decision node that splits the attributes.

Building a decision tree for test oracle required a set of pre-processed training data with attributes obtained in Table 10.

Table 10: Jimple predicates and attributes of triangle program

Attribute ID	Attribute	Predicate Node of Jimple CFG	Jimple Predicate
1	`a > = b + c`	`[5]`	`i0 > = $i3`
2	`b > = a + c`	`[7]`	`i1 > = $i4`
3	`c > = a + b`	`[9]`	`i2 > = $i5`
4	`a != b`	`[10]`	`i0 != i1`
5	`b != c`	`[11]`	`i1 != i2`
6	`a = b`	`[13]`	`i0 == i1`
7	`a = c`	`[14]`	`i0 == i2`
8	`b = c`	`[15]`	`i1 == i2`

The process of converting a set of auto-generated test inputs to the training set is called test data transformation (see Figure 7). Tabe 11 shows the organized training data converted from Table 3. The columns of Table 11 are eight attributes ID shown in Table 10. Each generated test input in in Table 3 is converted into corresponding rows under these columns. Similarly, their corresponding results are converted into cells under the *Result* columns. For example, for a generated test input `<7, 7, 7>` and its actual result 1, the transferred training data `(f, f, f, f, f, t, t, t, 1)` is shown in the first row of Table 11.

6 Empirical Study

The empirical study mainly focuses on generating test input and mining decision tree models for unit testing. Three programs are chosen to evaluate our approach, including `Triangle` program, the `NextDate` program [17], and the `Vending Machine` program [16]. The `NextDate` program simply computes the next date for a given date. The `Vending Machine` program is a classical job-interview question for software testing positions. The size of three programs (in Java and Jimple) and the number of predicates are listed in Table 12. The three programs cover a different number of Jimple predicates, which can be categorized into 3 groups, small (8), medium (19), and large (41). The number of Jimple predicates (e.g., 8 for `Triangle` problem) is consistent with the number of attributes used for mining decision trees (e.g., 8 attributes in Table Table 11). The table also includes the attributes used for the mining purpose. Although the subjects of the empirical study are small programs, our approach can be used for unit testing of individual methods of large programs.

Our study aims at answering the following questions:

1) What is the performance of the proposed approach?
2) How does the coverage criteria used for test input generation affect the model accuracy?
3) What is the fault detection capability of using the mined model for a test oracle?
4) The source code of the rule-based data generation algorithm can be found in https://github.com/frankwxu/Gannon-JVM. The empirical study results for questions 2 and 3 can be found in https://github.com/frankwxu/frankwxu-JSS_empirical_study.

Table 11: Transferred test data

Test input ID	**Attr 1**	**Attr 2**	**Attr 3**	**Attr 4**	**Attr 5**	**Attr 6**	**Attr 7**	**Attr 8**	***Result***
1	f	f	f	f	f	t	t	t	1
2	f	t	f	t	t	f	f	f	4
3	f	f	t	t	t	f	f	f	4
…									
13	f	f	f	t	t	f	t	f	2
14	f	f	f	t	t	f	f	t	2
15	f	f	f	t	f	f	f	f	3
16	t	f	f	t	t	f	f	f	4

Table 12: Program size

	Line of Code		Number of Predicates		
	Java	Jimple	Java	Jimple (allow duplications)	Attributes (No duplication)
Triangle	22	27	3	8	8
Next Date	48	51	9	19	19
Vending Machine	82	68	21	41	10

6.1 Performance of Test Input Generation

Our tool handles Java bytecode. It is difficult to compare our tool with tools that handle different languages. For example, Z3 [33] and PEX [40] generate test inputs for dot-net and CUTE [20] for C, respectively. EVOSUITE [12] can be a good candidate for performance comparison for a given goal. However, it is not suited for comparing the mutant detection rates against our approach as (1) EVOSUITE handles Java Source code not bytecode and (2) the detection rates depend on the number of the inputs generated for a given path. Therefore, instead of comparing mutant detection rates, we compare the execution time needed to generate test inputs for a given path. Table 13 shows execution time required to generate test inputs of three programs for a given path using rule-based search algorithm and EVOSUITE. These paths are grouped by the search goals shown in the table, i.e., triangle types. These experiments are run on a Dell PC with Intel Pentium Dual CPU E6750 @ 2.66 GHz, 2G RAM, 250GB HDD and 64-bit operating system. The results show that the average time to generate inputs is consistent with the number of predicates in the program for both approaches. The rule-based approach has a better performance as EVOSUITE uses genetic algorithms as the genetic algorithm often has a slow convergence problem.

Table 13: Performance of input generation

ID	Goal	Execution Time to Generate 100 Inputs (ms)	
		Rule-based	EVOSUITE
	Triangle		
1	Equilateral	255	883
2	Isosceles	167	454
3	Scalene	146	534
4	NotATriangle	121	125
	Next Date		
1	Normal day && 1st month	96	210
2	Last day && 1st month	168	464
3	Normal day	167	242
4	Last day of a normal month	198	122
5	Normal day in Dec.	114	343
6	Last day of a year	172	421
7	Normal day in Feb.	153	236
8	leap year	738	909
9	Non- leap year	779	1222
10	Incorrect days of a leap year	912	1234
	Vending Machine (Change, Dollar, Cents, Juice, Beer)		
1	(1,1,0,0.1)	534	675
2	(1,1,0,0,0)	553	758
3	(1,0,1,1,0)	523	904
4	(1,0,1,0,1)	495	964
5	(0,1,0,1,0)	277	578
6	(0,1,0,0,0)	339	732
7	(0,0,1,1,1)	285	456
8	(0,0,1,0,1)	310	634

6.1 Accuracy of Decision Tree In Terms of Coverage

The accuracy of a decision tree is defined as the number of correct classifications divided by the total number of classifications. We measured the accuracy of decision trees to determine whether we have had enough training datasets for maximizing the correctness of the model so that we can measure the fault detection capability of the model. In other words, more training data needs to be generated regarding test coverage if the accuracy of a decision tree is not reaching the expected accuracy, e.g., 100% in this empirical study. A high level of accuracy is expected because we manually determine the correct expected results for training data, assuming it contains no noises and therefore no pruning mechanism is needed, whereas normal training data for building decision trees naturally contains noises, which will be difficult to prune off by applying the built-in pruning mechanism. Due to our particular purpose of using accuracy, whether a test dataset is balanced, e.g., whether each test dataset (type 1, 2, 3, and 4 in `Triangle` example) has the equal percentage, will not directly determine the number of training data needed to generate a decision tree with a specified accuracy, but test coverage does. In the case of the 100% accuracy is produced due to false positive results, it will be detected and corrected when checking test oracle (described in the next subsection).

Our experiments use three test coverage criteria with respect to Jimple CFG: statement coverage, decision coverage, and unreduced decision coverage. Decision coverage generates minimum tests to cover the Jimple CFG predicates. Unreduced decision coverage generates two tests for each Jimple CFG predicate without reducing the duplicated tests. The protocol of our empirical study is as follows:

- Generate test inputs according to the coverage criteria. We initially chose test inputs that meet a given coverage criterion. The cover times, along with its corresponding number of tests, indicate how many tests we have used as training data in an experiment.
- Manually calculate the expected outputs of the generated test inputs.
- Convert the test inputs and expected outputs to training data.
- Generate a decision tree model use the C4.5 mining algorithm in WEKA data mining tool [41] which takes the training data as its inputs.
- Compute accuracy. The process repeats with incremented cover times until the accuracy of decision tree reaches 100%. Table 15 shows the model accuracy in terms of coverage criteria and cover times. For example, 8 test cases are needed for the minimum decision coverage of the Jimple CFG of the `Triangle` program. However, the generated decision tree only reaches 50% accuracy.

Table 14 shows that: (1) For different coverage criteria, the model accuracy increases as the number of cover times (the number of test cases) increases. For example, the accuracy of models generated based on decision coverage increases from 50% to 100%, 85.71% to 100%, 66.67% to 100% in `Triangle`, `NextDate`, and `Vending Machine` problems as the number of test cases increase, respectively. (2) Statement coverage has the fast performance to reach 100% accuracy. It only takes a total of 278 test cases to reach 100% in the three examples. Decision coverage and unreduced decision coverage need 472 and 524 test cases to reach 100% accuracy, respectively.

Note that the accuracy is not always consistent with the number of test cases. For example, in `NextDate`, the accuracy of the decision tree generated from decision coverage drops from 94.64% to 89.29% when the number of test cases increases from 56 to 84. The decrease in accuracy is caused by the existence of non-representative training data. The non-representative training data is treated as incorrectly classified data. For example, a new test case is added for the training data, the new case either follows the existing model or splits a tree node to form a new path. However, if the information gain, i.e., the criteria for splitting a tree node, calculated from C4.5 alogrithm is less than the splitting threshold, the training data will fail to split tree nodes unless more similar training data will be added. In this case, the newly added test cases fail to be classified into the appropriate groups, and hence, it decreases the accuracy of the decision tree model. In addition, the more predicates derived from Jimple code, the more cover times/test inputs needed to reach 100% accuracy. For example, on average, `Triangle` and `NextDate` have 8 and 19 predicates respectively. It only takes 6, 3, and 2 cover times to reach 100% accuracy, respectively, for `Triangle` problems, in terms of statement, decision, and unreduced decision coverage. On the contrary, it takes ten cover times to reach 100% accuracy for `NextDate`.

Table 14: Model accuracy (in percentage) in terms of coverage criteria and cover times

Cover Times	Statement		Decision		UnReduced Decision	
	Accuracy (%)	# of Tests	Accuracy (%)	# of Tests	Accuracy (%)	# of Tests
Triangle						
1	50.00	4	50.00	8	68.75	16
2	50.00	8	87.50	16	100.00	32
3	66.67	12	100.00	24	x	x
4	75.00	16	x	x	x	x
5	90.00	20	x	x	x	x
6	100.00	24	x	x	x	x
Next Date						
1	80.00	10	85.71	28	86.84	38
2	95.00	20	*94.64*	56	88.16	76
3	93.33	30	*89.29*	84	95.61	114
4	95.00	40	98.21	112	97.37	152
5	96.00	50	95.00	140	97.90	190
6	96.67	60	96.43	168	98.25	228
7	98.57	70	96.43	196	99.62	266
8	97.50	80	98.67	224	99.34	304
9	98.89	90	99.20	252	99.70	342
10	100.00	100	100.00	280	100.00	340
Vending Machine						
1	72.72	22	66.67	24	94.74	76
2	90.91	44	87.50	48	100.00	152
3	84.85	66	94.44	72	x	x
4	88.64	88	91.67	96	x	x
5	92.73	110	95.00	120	x	x
6	96.97	132	98.61	144	x	x
7	100.00	154	100.00	168	x	x
Total Tests for Reaching 100% Accuracy						
278		472		524		

Note: Accuracy is omitted (marked as "x") after reaching 100%

6.3 Fault Detection Using Decision Trees

Traditionally, fault detection capability refers to the percentage of the mutants killed by the given tests. Mutants are software faults introduced by programmers due to small syntactic errors. For example, the expression `a && b` is incorrectly written as `a || b`. A mutant is said to be killed if the actual result of a test is different from its expected result. In our approach, a mismatch is caused by either the fault in a mutant or the incorrectly generated decision tree. We will discuss these two scenarios separately. The process for analyzing the fault detection capability of our approach is as follows:

- Select fault types and mutation operators for the subject programs,
- Create mutants by inserting one fault at a time,
- Build a decision tree with 100% accuracy by applying our approach using the faulty version of the UUT,
- Use the rest of test inputs as validation data and decision trees to find mismatches, and
- Examine mismatches.

Our empirical study chose six common mutation operators [35] including arithmetic operator replacement, arithmetic operator insertion, relational operator replacement, etc. They are shown in Table 15.

Table 15: Selected mutation types for empirical study

Mutation Operator			Examples	
Category	ID	Type	Original	Replaced
Arithmetic Operations	1	Arithmetic Operator Replacement	a + b	a - b
	2	Arithmetic Operator Insertion	b + c	-b + c
Relations	3	Relational Operator Replacement	a != b	a == b
Conditions	4	Conditional Operator Replacement	(a==b) && (b==c)	(a==b) || (b==c)
Constants	5	Constant Value Modification	s = a	s = b
Return Values	6	Return Value Modification	return s	return s’

Table 16: The number of Tests to find the first mismatch using decision tree generated by different coverage

Mutation ID	# of Mutants	# of Tests Executed			Oracle Results					
					# Mutants Discovered			# Faults in Models		
		S	D	U	S	D	U	S	D	U
Triangle Problem										
1	3	4	4	4	3	3	3	0	0	0
2	4	16	16	16	3	3	3	1	1	1
3	3	5	5	5	3	3	3	0	0	0
4	1	6	6	6	1	1	1	0	0	0
5	3	6	6	6	3	3	3	0	0	0
6	3	4	4	4	3	3	3	0	0	0
Next Date Problem										
1	3	13	15	15	2	3	3	1	0	0
2	6	27	37	37	4	6	6	2	0	0
3	4	22	19	24	4	3	4	0	1	0
4	3	26	18	27	3	2	3	0	1	0
5	2	6	6	6	2	2	2	0	0	0
Vending Machine										
2	3	30	30	30	3	3	3	0	0	0
3	4	45	45	45	4	4	4	0	0	0
5	4	39	39	39	4	4	4	0	0	0
6	4	41	41	41	4	4	4	0	0	0
Total	50	886			142			8		

** S: Statement Coverage D: Decision Coverage U: Unreduce Decision Coverage*

The results are shown in Table 16. It records the number of test cases needed to find the first mismatch using the decision trees generated by statement coverage, decision coverage, and unreduced coverage. In Table 16 our experiments are grouped based on mutation operators. For example, we have created four mutants of mutation operator 2 and implanted each of them into the `Triangle` program one at a time. To detect mismatches between the actual and expected results caused by these mutants, 16 validation test inputs are used for each coverage, statement coverage (**S**), decision coverage (**D**), and unreduced decision coverage (**U**). There are two types of oracle results shown in the table due to the causes of mismatches: the number of mutants discovered and the number of faults in the model. The mismatches caused by mutants planted in the source code are counted as the number mutants discovered. The mismatches caused by the incorrect decision tree models are counted as the number of faults in models. After examining the four mismatches, we found that three of them are caused by the mutants and the remaining mismatch is caused by the incorrect decision tree. Consider executing a test case with the input <`2, 2, 7`> using a faulty version of the `Triangle` program that results in "`NotATriangle`". The corresponding output in the decision tree is "`Isosceles`". It implies the mismatch is caused by the model.

In summary, a total of 886 test cases is executed to exercise 150 mutants (50 mutants for S, D, and U, respectively) of the `Triangle`, `NextDate`, and `Vending Machine` programs. 142 out of 150 (94.67%) mutants are killed. The remaining eight mismatches are due to the incorrect decision tree models. Note that the fault detection capabilities with respect to the three coverage criteria of the `Triangle` programs are the same, i.e., 94%, because they result in the same model. The fault detection capabilities in `NextDate` are 83%, 89%, and 100% for statement coverage, decision coverage, and unreduced decision coverage, respectively. In addition, the fault detection capabilities with respect to all three coverage criteria of the `Vending Machine` program are 100% because the input domains of the program are binary numbers and thus these test cases generated with respect to these three coverage criteria covers all possible scenarios. Specifically, there are a total of 32 possible inputs with five binary parameters. For example, the `Vending Machine` only takes 50 cents or a one dollar bill, and the dispense button either has being pushed or not being pushed. The model generated using unreduced decision coverage has the best fault detection capability (i.e., 98%=49/50).

6.4 Using Decision Trees as Test Oracle in Practice

Using decision trees as test oracle is a reasonable approach for testing purpose based on the following facts:

- The control structure is a key element of any program languages.
- Control structure uses predicates to control executing flow.
- A decision tree is built from a set of predicates and a training dataset.
- The decision tree model represents the knowledge of the control structure of UUT.

In practice, building the decision tree model and using the model to detect faults in UUT is an iterative process. The following guidelines can be used to detect faults in UUT and inaccurate decision tree models.

- Step 1: Choosing a UUT contains a number of predicates to build a decision tree model. A program without control structure does not fit into the proposed approach due to lacking of attributes for building the decision tree model.
- Step 2: Generate test input automatically based on test coverage, i.e., the decision coverage.
- Step 3: Build a decision tree based on the proposed approach. Do not use any pruning mechanisms as we expect the decision tree can be validated by all training and validation data. If the accuracy of the model is not reaching 100%, developers need to check which training data causes the mismatch. This may be caused by either incorrect expected results determined by developers manually or the faults in UUT. Otherwise, developers need to repeat step 2 to improve the accuracy of the model by generating more training datasets. The process ends when all generated datasets fit the decision tree model with 100% accuracy.

Four tasks determine the efficiency of the proposed approach: 1) the time to generate test inputs, 2) the time for preparing training data to build the decision tree model, 3) the time spent to construct the decision tree model, 4) the time for checking test oracle, and 5) the time for regenerating/fixing models. All tasks except task 2 can be automated and therefore are insignificant for the whole process. Task 2 is determined by the number of test inputs which needs multiples of the time for finding their expected results. Assume that the efforts developers devoted to finding the expected result with each given input is the same, the efficiency of the proposed approach is determined by the number of the test inputs generated in terms of testing coverage. Figure 8describes the efficiency of the approach in terms of testing coverage. The x-axis represents the percentage of accuracy and the y-axis represents the efforts made for generating decision models. The figure indicates:

- Overall, approximately four times of efforts are needed to increase the accuracy from 90% to 100% comparing from 0% to 90%.
- More attributes used for building decision tree models, the more efforts needed to build the models to reach 100% accuracy. The figure shows that `NextDate` program with 19 attributes needs twice and 10 times efforts than `Vending Machine` (10 attributes) and `Triangle` (8

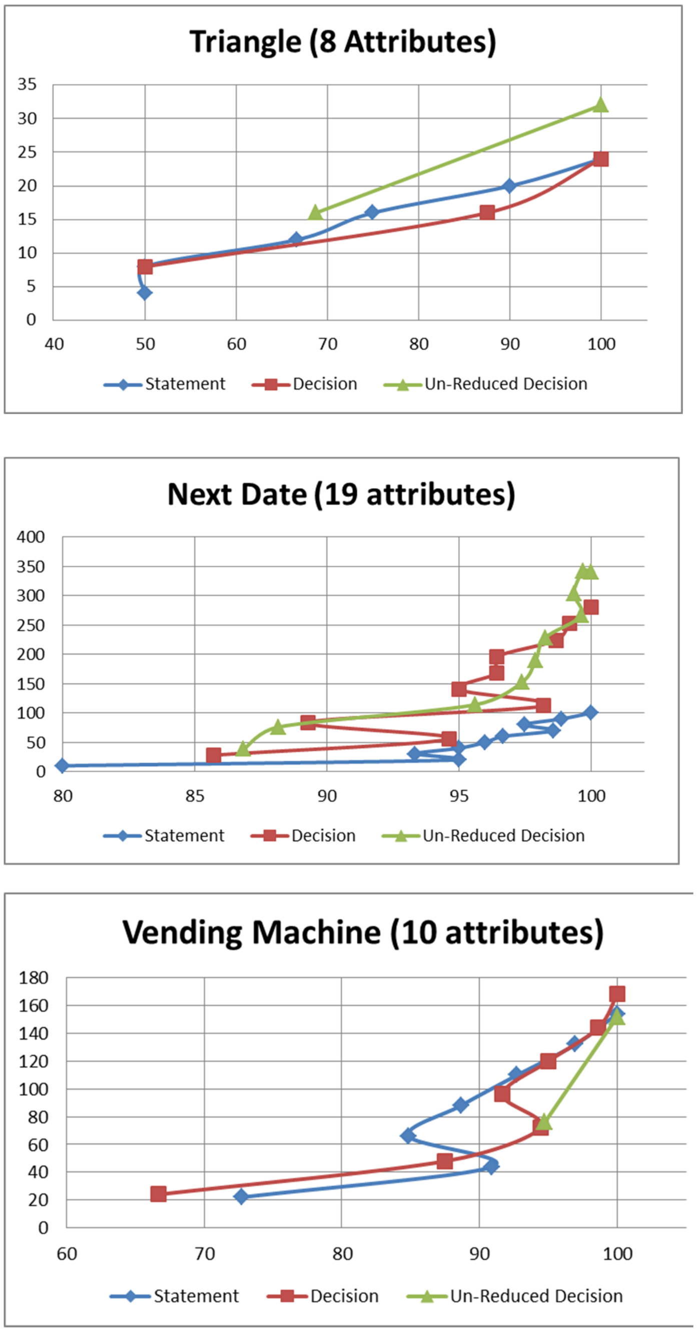


Figure 1: Efficiency of Approach in Terms of Coverage and the Number of Attributes

attributes), respectively.

- Unreduced decision coverage has the best fault detectability based on our previous empirical study; however, more efforts are needed to generate the decision tree models for test oracle.

6.3 Threats to Validity

Several factors can jeopardize the internal and external validity of the empirical study. First, the subject programs are all small in term of lines of code. The main reason is that the proposed approach focuses on unit testing. Although in principle, it can be applied to the unit-testing of individual methods of large object-oriented programs, complex interactions between methods in real-world software need to be considered. Second, the currently proposed approach deals with basic data types, such as integer, char, Boolean, and string, but not general object types. Dealing with general object types would require a major improvement to the proposed approach. Third, the proposed approach depends on the decision tree technique, which may not be suitable for all programs. Finally, the evaluation of fault detection capability is based on the mutants created by six mutation operators. They do not necessarily cover all possible bugs in real-world software.

7 Related Work

Various search-based techniques have been applied to automated generation of test inputs. Existing research falls into two categories: (a) the local optimal solution that is optimal within a neighboring set of solutions and (b) the global optimal solution that produces either maximal or minimal. Typical metaheuristic search algorithms using local optimized solutions are Hill Climbing (HC) and Simulating Annealing (SA) [28]. Both algorithms are similar, however, SA allows for less restricted movement around the search space. As both HC and SA are local search approaches, their performance to generate desired test inputs is usually lower than global optimal approach, e.g., genetic algorithm (GA) [15, 37, 43]. Harman *et al.* [29] conducted a theoretical and empirical study, comparing random search (RS), HC and GA on a number of test projects. The study showed the evolutionary algorithms are suitable in many situations when it comes to the generation of input test data for structural testing, whereas in some cases simpler search techniques perform surprisingly well, and are able to surpass evolutionary algorithms. Alternative Variable Method (AVM) is mentioned by Korel [18] and is another algorithm based on local search. Arcuri [3] analyzed the runtime of three search algorithms (RS, HC and AVM) on the test data generation; the empirical study proves AVM has the best time performance among three algorithms. Recently, Fraser and Arcuri [12] have proposed a new GA-based approach, named whole test suite generation, for test data generation by evolving all the test cases in a test suite at the same time, and the fitness function considers all the testing goals simultaneously. Global optimization is difficult for metaheuristic search algorithms as these algorithms do not take structure of code into consideration. In contrast, global optimization analyzes static or dynamic of UUT source code to understand the data relationships globally to improve the search performance. Gotlieb *et al.* [4] transformed a procedure into a constraint system by using Static Single Assignment form and control dependencies. Their approach analyzes global constraints and thus the reduction of the number of trials required for the generation of test data. McMinn *et al.* [30] have proposed a static dependence analysis derived from program slicing that can be used to support search space reduction. Korel [18-19] has applied dynamic data-flow analysis to determine those input variables responsible for the undesirable program behavior, significantly increasing the speed of the search process. The data-flow analysis is based on data define-use or last define-use relations. Note that many approaches use symbolic evaluation [13, 34, 45] to derive test data, however, they require complex algebraic manipulation. Different from these approaches, the rule-based testing data generation in this paper is a hybrid approach that utilizes metaheuristic RBA to replace fitness functions and takes the advantage of static analysis of bytecode to compute global optimized test inputs. It is based on the internal structure and actual execution of the UUT. In addition, utilizing RBA for testing data generation aligns with the process of mining test oracle as RBA provides data search guidelines based on predicates and the same predicates will be used for mining test oracle.

Mining behavioral models from test cases has gained much attention in recent years. Existing research falls into two categories: (a) detecting invariants over data values or event sequences, and (b) generating state transition models from execution traces. Techniques for detecting invariants over data values or sequences extract specifications in the forms of pre-conditions and post-conditions [9, 11] and algebraic axioms[14]. Values assigned to variables at specific program points can provide important information to understand and analyze system executions. Daikon [9] is an implementation of dynamic detection of likely invariants. It runs a program, observes the values that the program computes, and then reports properties that were true over the observed executions. Hankel et al. [14] probe Java classes by invoking them on dynamically generated tests and capture the information observed during their execution as algebraic axioms. Algebraic specifications relate the meaning of sequences of code operations, such as `pop(push(x,stack))= stack`. Although it discovers many useful axioms when applied to container classes, the tool is not complete or correct from a formal perspective. Techniques for generating state transition models of execution traces focus on extracting finite-state machines (FSMs) and extended FSMs with parameters. They generalize the observed executions by merging states that exhibit similar behavior. Most existing approaches are based on the kTail algorithm [6] or its variants. kTail merges states based on the similarity of the next *k* invocations in the trace. GK-tail [26] generates extended FSMs from positive invocation sequences and uses Daikon to infer from values the constraints on state transitions. Walkinshaw and Bogdanov [42] and Lo et al. [24] augment the inference of FSMs with a steering mechanism based on temporal properties. Different from the above work, our approach generates test inputs from Java bytecode and builds a decision tree from a subset of test inputs, and derives test oracles for the remaining or new test inputs. Last et al. [22] used info-fuzzy networks to represent the discovered input-output relations, where rules are learned by computing the weight of each node in the networks. Statistical attributes comparison [27] and support vector machines [44] have also been used to classify data, analyze data, and recognize patterns. Statistical attributes comparison check the statistical properties of inputs and outputs, such as statistical mean, variance, etc. The test oracle is to verify property consistency of the expected and actual results. The method of support vector machines can approximate the function from inputs to outputs by computing the maximum-margin hyperplane that divides training points. Different from the above work, our approach builds a decision tree from the test inputs automatically generated from Java bytecode using coverage criteria.

Several studies have been conducted to create different models explicitly as test oracles for fault detection. Lo et al. [23] mines a set of discriminative features capturing repetitive series of events from program execution traces. These features are then used to train a classifier to detect failures. Similarly, Milea et al. [31] have proposed a mining approach to build a normal behavior model as a test oracle to monitor malicious behavior for windows. The model of normal behavior is based on a rich set of discriminators such as minimal infrequent and maximal frequent iterative patterns of system calls, and relative entropy between distributions of system calls. Bowring et al. [7] models program executions as Markov models, and a clustering method for Markov models that aggregates multiple program executions into effective behavior classifiers. Different from these approaches, [36] Pacheco and Ernst build an operational model from observations of the software running properly. The operation model includes object invariants and properties. The object invariants are the conditions that hold on entry and exit of all public methods. Our approach is the complement of theirs, which generates and classifies inputs based on the internal structure of the UUT. Briand [8] has proposed the use of machine learning techniques - including decision trees - for the test oracle problem. The major difference between our approach and his are twofold: first, the information used for building decision tree models is different. Briand indicates that without some additional guidance, using only the test case input and output values is insufficient to build models. The decision tree model he has proposed is constructed from abstract test cases or software requirements. We construct the decision tree models from auto-generated test inputs and relying on the structural of UUT, i.e., predicates. Second, the decision tree models represent different knowledge. Briand's decision tree models represent failure conditions. Our models represent all the behavior of the training dataset. The models need to be verified when a validation input does not follow the rules in the model. Recent studies on mining behavioral models [5, 21, 25] have discovered valuable insight of how the model relates to the underlying system, which provide solid foundation for mining test oracle.

8 Conclusions

We have presented an approach to mining test oracles of test inputs that are automatically generated from Java bytecode. Our empirical study indicates that fewer test cases are needed for building decision trees when statement coverage is used to generate test inputs, while the same fault detection capability can be achieved. Using the mined test oracles, 94.67% mutants are killed by the generated test inputs. The model generated using unreduced decision coverage has the best fault detection capability (i.e., 98%).

Our current work uses decision trees to represent the system behaviors. Decision tree is closely related to rule induction [39]. Each path from the root to a leaf can be transformed into a rule. The rule can be constructed simply by conjoining the tests along the path to form the antecedent part. The leaf of the tree is the inductive value. For example, one of the paths in the figure (inside Table 3) can be transformed into the rule: "For a given input <`a`, `b`, `c`> to test a triangle, if `a` equals `b` and `b` equals `c`, then the type of the triangle is `Equilateral (Type 1)`". Thus, our approach is effective for rule-based systems. Currently, our study used only small programs with integer variables. Our future work will investigate the scalability of our approach and deal with complex data types. Another plan is to utilize ASM [10] for test inputs generation and test oracle mining due to its high performance and popularity. ASM is a very small and very fast Java bytecode manipulation framework supported by Open Solutions Alliance.

References

[1] W. Afzal, T. Richard, and F. Robert, "A Systematic Review of Search-Based Testing for Non-Functional System Properties," *Information and Software Technology,* 51(6):957-976, 2009.

[2] S. Ali, L. Briand, H. Hemmati, and R. Panesar-Walawege, "A Systematic Review of the Application and Empirical Investigation of Search-Based Test-Case Generation," *IEEE Transactions on Software Engineering,* 36(6):742-762, 2010.

[3] A. Arcuri, "Full Theoretical Runtime Analysis of Alternating Variable Method on the Triangle Classification Problem," Search Based Software Engineering, 2009 1st International Symposium, 2009.

[4] G. Arnaud, B. Bernard, and R. Michel, "Automatic Test Data Generation Using Constraint Solving Techniques," *ACM SIGSOFT Software Engineering Notes,* 23(2):53-62, 1998.

[5] I. Beschastnikh, Y. Brun, J. Abrahamson, M. D. Ernst, and A. Krishnamurthy, "Unifying FSM-Inference Algorithms through Declarative Specification," 35th International Conference on Software Engineering, San Francisco, CA, 2013.

[6] A. W. Biermann and J. A. Feldman, "On the Synthesis of Finite State Machines from Samples of their Behavior," *IEEE Transactions on Computer,* 21:592-597, 1972.

[7] J. F. Bowring, J. M. Rehg, and M. J. Harrold, "Active Learning for Automatic Classification of Software Behavior," International Symposium on Software Testing and Analysis, Boston, Massachusetts, 2004.

[8] L. C. Briand, "Novel Applications of Machine Learning in Software Testing," 8th International Conference on Quality Software, Oxford, UK, 2008.

[9] J. Cockrell, M. D. Ernst and W. G. Griswold, "Dynamically Discovering Likely Program Invariants to Support Program Evolution," *IEEE Trans. on Software Engineering,* 27(2):99-123, 2001.

[10] O. Consortium, "ASM," [Online]. Available: http://asm.ow2.org/. [Accessed 23-08-2013].

[11] M. D. Ernst, J. H. Perkins, P. J. Guo, S. McCamant, C. Pacheco, M. S. Tschantz, and C. Xiao, "The Daikon System for Dynamic Detection of Likely Invariants," *Science of Computer Programming,* 69(1-3):35-45, 2007.

[12] G. Fraser and A. Arcuri, "Whole Test Suite Generation," *IEEE Transactions on Software Engineering,* 39(2):276-291, 2013.

[13] M. Harman, A. I. Baars, Y. Hassoun, K. Lakhotia, P. McMinn, P. Tonella, and T. E. J. Vos, "Symbolic Search-Based Testing," The 26th IEEE/ACM International Conference on Automated Software Engineering, Lawrence, KS, 2011.

[14] J. Henkel, C. Reichenbach, and A. Diwan, "Discovering Documentation for Java Container Classes," *IEEE Trans. on Software Engineering,* 33(8):526-543, 2007.

[15] J. Holland, *Adaptation in Natural and Artificial Systems*, Ann Arbor, University of Michigan Press, 1975.

[16] "How Would You Test a Vending Machine?," 1-12-2011, [Online]: Available: http://www.softwaretestingquestions.net/category/testing-in-the-wild/. [Accessed 24-1-2013].

[17] P. C. Jorgensen, Software Testing: *A Craftman's Approach*, 3rd Ed., Auerbach Publications, 2008.

[18] B. Korel, "Automated Software Test Data Generation," *IEEE Transactions on Software Engineering,* 16:870-879, 1990.

[19] B. Korel, "Automated Test Data Generation for Programs with Procedures," ACM SIGSOFT International Symposium on Software Testing and Analysis, New York, NY, USA, 1996.

[20] S. Koushik, D. Marinov, and G. Agha, "CUTE: A Concolic Unit Testing Engine for C," 10th European Software Engineering Conf. held jointly with 13th ACM SIGSOFT Int. Symposium on Foundations of Software Engineering, Lisbon, Portugal, 2005.

[21] S. Kumar, S.-C. Khoo, A. Roychoudhury, and D. Lo, "Inferring Class Level Specifications for Distributed Systems," 34th International Conference on Software Engineering, Zurich, Switzerland, 2012.

[22] M. Last, M. Friedman, and A. Kandel, "Using Data Mining for Automated Software Testing," *International Journal of Software Engineering,* 14(4):369-393, 2004.

[23] D. Lo, H. Cheng, J. Han, S.-C. Khoo, and C. Sun, "Classification of Software Behaviors for Failure Detection: A Discriminative Pattern Mining Approach," 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Paris, France, 2009.

[24] D. Lo, L. Mariani and M. Pezzè, "Automatic Steering of

Behavioral Model Inference," The 7th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software EngineeringAmsterdam, The Netherland, pp. 345-354, 2009.

[25] D. Lo, L. Mariani and M. Santoro, "Learning Extended FSA from Software: An Empirical Assessment," *Journal of Systems and Software,* 85(9):2063-2076, 2012.

[26] D. Lorenzoli, L. Mariani, and M. Pezzè, "Automatic Generation of Software Behavioral Models," The 30th International Conference on Software Engineering, Leipzig, pp. 501-510, 2008.

[27] J. Mayer and R. Guderlei, "Test Oracles Using Statistical Methods," *Proc. of the First Int'l Workshop on Software, Springer*, pp. 179-189, 2004.

[28] P. McMinn, "Search-Based Software Test Data Generation: A Survey," *Software Testing, Verification and Reliability,* 14(2):105-156, 2004.

[29] M. McMinn and P. Harman, "A Theoretical and Empirical Study of Search-Based Testing: Local, Global, and Hybrid Search," *IEEE Transactions on Software Engineering,* 36(2):226-247, March-April 2010.

[30] P. McMinn, M. Harman, K. Lakhotia, Y. Hassoun, and J. Wegene, "Input Domain Reduction through Irrelevant Variable Removal and Its Effect on Local, Global, and Hybrid Search-Based Structural Test Data Generation," *IEEE Transactions on Software Engineering,* 38(2):453-477, 2012.

[31] N. A. Milea, S.-C. Khoo, D. Lo, and C. Pop, "NORT: Runtime Anomaly-Based Monitoring of Malicious Behavior for Windows," Runtime Verification, Lecture Notes in Computer Science, pp. 115-130, 2012.

[32] B. M. E. Moret, "Decision Trees and Diagrams," *Computer Survey,* 14(4):593-623, December 1982.

[33] L. D. Moura and N. Bjørner, "Z3: An Efficient SMT Solver," 14th International Conference on Tools and Algorithms for the Construction and Analysis of Systems, Budapest, Hungary, 2008.

[34] J. Offutt, Z. Jin, and J. Pan, "The Dynamic Domain Reduction Approach to Test Data Generation," *Software-Practice and Experience,* 29(2):167-193, 1999.

[35] A. J. Offutt, A. Lee, G. Rothermel, R. Untch, and C. Zapf, "An Experimental Determination of Sufficient Mutation Operators," *ACM Transactions on Software Engineering Methodology,* 5(2):99-118, April 1996.

[36] C. Pacheco and M. D. Ernst, "Eclat: Automatic Generation and Classification of Test Inputs," 19th European conference on Object-Oriented Programming, Glasgow, UK, 2005.

[37] R. P. Pargas, M. J. Harrold, and R. R. Peck, "Test-Data Generation Using Genetic Algorithms," *Software Testing, Verification And Reliability,* 9:263-282, 1999.

[38] J. R. Quinlan, *C4.5: Programs for Machine Learning*, San Francisco: Morgan Kaufmann Publishers, 1993.

[39] L. Rokach and O. Maimon, "Decision Trees," *The Data Mining and Knowledge Discovery Handbook,* pp. 165-192, 2005.

[40] N. Tilmann and J. d. Halleux, "Pex — White Box Test Generation for .Net," International Conference on Tests And Proofs, Prato, Italy, 2008.

[41] U. O. Waikato, "Weka 3: Data Mining Software in Java," University of Waikato, [Online]. Available: http://www.cs.waikato.ac.nz/ml/weka/. [Accessed 10 Jan 2013].

[42] N. Walkinshaw and K. Bogdanov, "Inferring Finite-State Models with Temporal Constraints," The 23rd IEEE/ACM International Conference on Automated Software Engineering, pp. 248-257, L'Aquila, Italy, 2008.

[43] J. Wegener, H. Sthame, B. F. Jones, and D. E. Eyres, "Testing Real-Time Systems using Genetic Algorithms," *Software Quality Journal,* 15(3):127-135, 1997.

[44] M. Ye, B. Feng, L. Zhu, and Y. Lin, "Using Wavelet Support Vector Machines to Generate Expected Outputs," 5th IEEE International Conference on Cognitive Informatics, Beijing, China, 2006.

[45] R. N. Zaeem and K. Sarfraz, "Test Input Generation Using Dynamic Programming," 20th International Symposium on the Foundations of Software Engineering, Research Triangle Park, NC, 2012.

Weifeng Xu is an Associate Professor in the Department of Computer Science at Bowie State University. He received his B.S. and M.S. degrees in Computer Science from Southeast Missouri State University and Towson University at Maryland, respectively. He also received his Ph.D. in Software Engineering from North Dakota Statement University. His current research efforts focus on mining software engineering data, software testing, and software security. He is a senior member of the IEEE.

Tao Ding received a Bachelor Degree in Software Engineering from China, and Master Degree in Computer and Information Science from Gannon University, Pennsylvania, US. She is currently a Ph.D. student in Department of Information Systems at the University of Maryland, Baltimore County. Her research interests include natural language processing, machine learning and software engineering.

Dianxiang Xu received the B.S., M.S., and Ph.D. degrees in Computer Science from Nanjing University, China. He is a Professor of Computer Science at Boise State University, USA. His prior teaching and research experience has included positions at Dakota State University, North Dakota State University, Texas A&M University, Florida International University, and Nanjing University. His research interests include software security and safety, software testing, applied formal methods, and computer forensics. He has published more than 100 peer-reviewed papers in international journals and conference proceedings. He is a senior member of the IEEE.

Omar El Ariss received his B.S. and M.S. degrees in Computer Science from the Lebanese American University, Beirut, Lebanon in 2001 and 2005, respectively. He got his Ph.D. degree in Computer Science from North Dakota State University in 2011. He is currently an Assistant Professor in the Department of Computer Science at The Pennsylvania State University-Capital College. His research interests are in the areas of software verification & validation, software safety & security, and natural language processing.

Predicting Fair Housing Market Value: A Machine Learning Investigation[1]

Timothy Oladunni* and Sharad Sharma*
Bowie State University, Bowie, Maryland 20715 USA

Abstract

The real estate market plays a major role in the economy of most developed nations, particularly the United States, thus a study of its market value prediction using machine learning algorithms is very important. This paper describes a two phase research into the predictability of the fair market value of real estate properties. The first phase concentrated on building a predictive real estate listings application software using an MVC architecture and linear regression. Residential real estate datasets extracted from Howard County, Maryland were used for the experiment. Performance evaluation was measured using r-squared with a performance outcome of 0.92. The second phase of the study experimented the model with four different learning algorithms; K-NN, improved neural network, polynomial regression and linear regression. Experiments were conducted with datasets of residential real estate listings from Baltimore and Montgomery counties. Spearman's rho was used for the performance comparisons of the algorithms. Neural network performed consistently better in both counties with a Spearman's rho value of 0.836 and 0.739 in Montgomery County and Baltimore County respectively. Statistical analysis was done using scatter plot, covariance matrix, correlation matrix and test of significance. Statistical analysis showed that the predictive parameters of real estate properties are not the same in all counties.

Key Words: Real estate, linear regression, neural network, K-NN, polynomial regression, housing prices prediction, machine learning, MVC, UML.

1 Introduction

The real estate market is very unstable and complicated at the moment in the United States. Homeowners are always eager to know the market value of their properties to enable them make appropriate and timely decision on when to sell or refinance. Many buyers of real estate properties always stay on the side line waiting for 'buyers' market' – a time when they can buy properties at moderate prices. Property appraisals and real estate agents are obligated by local authority to give professional advice on the estimated market value of a property at a particular jurisdiction. Some local governments get most of their income from the property taxes which are directly related to the value of properties. Banks and other financial institutions are also concerned about making prudent financial decision to mortgage applicants. On the side lines are investors who want to jump on any 'good' deal and make a flip. In fact, almost everyone in the public is affected by the real estate market.

All key players should be able to have a satisfactory answer to the question "what is the fair market value of this property?" Unfortunately, the present Multiple Listing Services (MLS) does not have a satisfactory answer to this question. Added to the problem is the fact that most home buyers in the United States do not request for an appraisal report until they have a ratified contract. Appraisal report is the estimated value of a property as determined by a licensed property appraiser. Buyers most of the time sign a contract on a property they are not sure of its value, often prices are inflated by real estate agents to the advantage of their sellers. The public is left to the professional view of licensed property appraisers, in some cases, they provide a biased assessment. Therefore, a research into a predictive model for the market value of properties is very important, crucial and vital to the economic decisions of buyers, sellers, real estate agents, property appraisals, financial institutions and the government. Various approaches have been used to generate prediction, however, our approach used machine learning. A machine learning approach to the prediction of market values makes a machine to learn from training data and predict the value of a property [7, 42].

With an ever increasing amount of data generated mostly because of the ubiquitous nature of computers and smart-phone devices in the world coupled with the availability of internet to everyone, the World Wide Web (WWW) has provided data in this decade more than the previous decades. Thus, the world is overwhelmed, saturated and packed with data more than ever anticipated. Added to this is the availability of inexpensive disks and online storage that takes data to a new level. While the variety, velocity and volume of data is magnanimous and its contents are complex, yet lying hidden are useful and vital information for the benefit of the public [22]. This led to the emergence of a quintessential tool like machine learning that can sniff through these large piles of data, learn its pattern and provide invaluable information. Machine learning enables a computer to 'learn', process and provide intelligent information

[1] Extended paper from proceeding at the ISCA 24th International Conference on Software Engineering and Data Engineering (SEDE 2015), San Diego, CA, October 12-14, 2015.

* Department of Computer Science. Email: oladunnit0423 @students.bowiestate.edu, ssharma@bowiestate.edu.

just as a human being [45]. Therefore, in recent time, the use of machine learning has been very popular; many industries have found it very useful, effective and suitable in maintaining a competitive edge in their decision making.

Wang et al. [36] discussed the application of machine learning algorithms in building an adaptive control system for bipedal robot control. Some researchers also argued that machine learning algorithms have the capability of adapting to the dynamic behavior of wireless sensor networks (WSNs) [2]. Nair et al., [29] compared the performance of different learning algorithms in diagnosing patients with arthritis. The outcome of the experiment showed that least-square kernel (LSK) had the highest performance classification. Muneki et al., proposed the use of belief propagation and linear response approximation learning algorithms for Boltzmann machines. Numerical experiments were conducted in validating the proposed scheme [47]. King et al., argued that machine learning algorithms are very efficient in selecting power control systems in a network [21]. Hsu et al., implemented machine learning algorithms in detecting malicious codes [14]. A bioinspired spike-based learning algorithm was applied to a liquid-state machine (LSM). The proposed scheme performed better than the hidden Markov-model in recognizing isolated words [50]. Gastaldo et al., investigated the implementation of machine learning algorithms on digital hardware [8]. Machine learning algorithms have been proposed for image resolution on mobile devices [3]. Some researchers proposed the use of embedded machine learning algorithms in analyzing medical-sensor signals [24].

Considering the rapid development and success in the application of machine learning to different aspects of the society, a vast majority of scientists are of the opinion that machine learning will surpass human intelligence in the future. They pointed out that the ability of a machine could be increased beyond the cognitive reasoning of humans. However, another school of thought is of the contrary view that since machines are man-made, the designer is still responsible to stipulate its capability, usability, robustness and performance; therefore, it is preposterous, illogical and impossible to expect a machine to surpass or eclipse the intelligence of its designer in any case or form. In either side of the aisle, the concession is that in the nearest future, the performance and capability of a machine will improve and be capable of competing with human intelligence [49].

We will discuss related works about house price prediction in the next section, and predictive software application in section three. In section four we will discuss the result and analysis, the implementation of a predictive model will be in section five. A comparison of four predictive models will be investigated in section six while section seven will be about our future work. We will conclude the paper in section eight with necessary remarks.

2 Literature Review and Definitions

2.1 Related Works

The dominant role that real estate plays in a modern economy has attracted many research works and studies in predicting the values of real estate properties. Researchers have proposed different methodologies in predicting, forecasting or estimating the prices of real estate properties. Some researchers used statistical tools like Z- score, correlation, regression and standard deviation while others have used machine learning algorithms like neural network, linear regression, logistic regression, Gaussian mixture model and support vector regression among others.

Unlike other areas of investment, real estate investment risk is more complex, complicated and volatile with a very high risk factor. Therefore, a detailed scientific analysis and prediction are necessary for potential risks to home buyers, sellers and investors. Li et al, argued that risk analysis and prediction is crucial for effective risk prevention and management in real estate investment. The researchers used a support vector machine (SVM) modeling algorithm approach to investigate and predict real estate investment risk [26].

Since the housing bubble and its subsequent burst which was blamed on sub-prime mortgage in the USA, a significant part of the US public have developed cold feet and lost confidence in the industry with a continued worry about the risk of investment. Thus, trading volumes have taken a nose dive, government budgets that depend on revenues from real estate have been affected and the equities of homes have disappeared. Considering the associated risks, banks and other financial institutions have increased the credit requirements and scrutiny of potential borrowers. BP neural network has been proposed as capable of assessing risk in real estate transactions. The experiments demonstrated that BP neural network was capable and suitable for a reliable and detailed assessment and management of real estate risks [40, 46].

Da Ying Li et al proposed the use of support vector regression (SVR) to predict real estate prices in China. They examined the application of support vector regression in real estate price prediction. To achieve a reasonable outcome, five variables were selected as the input variables and real estate price was used as the predictive variable. The outcome of the experiment was based on the mean absolute percentage error (MAPE), mean absolute error (MAE), and the root squared error RMSE. The experiment showed that the SVR model performed better than Back Propagation Neural Network model, therefore he concluded that SVR based approach seemed to be a better and more efficient tool to forecast real estate prices [25]. Kang and Xiang [19] reconstructed a new Z-Score predictive model for Chinese real estate financial prediction by re-evaluating all relevant, necessary and sensitive financial indicators. The researchers argued that the Chinese real estate market is unstable, volatile and chaotic. These factors according to the researchers have a direct effect on a long time prediction. Admitting these short comings, the proposed algorithm was capable of predicting a “two-years-ahead” risk prediction. With “two-years-ahead”, according to the researchers, the model can help investors and buyers to timely avoid risks associated with unexpected financial crisis, chaos and confusions associated with the Chinese real estate industries.

A combination of rough set (RS) and support vector machine (SVM) algorithms approach was proposed to determine a new way to estimate and predict the value of real estate properties by applying hedonic price solution [37, 38]. Zheng and Bing referred to hedonic price model as an algorithm in which several regression analyses were conducted and analyzed. The idea was to select a rough set to reduce numbers of variables which helps to obtain a minimal set of features and a reduced dimension of the input space of the support vector machine. With a reduced data as the input space of SVM, the convergence speed and the predicting capability of the algorithm improved [51].

Some researchers compared the performance of back-propagation neural network (BPN), fuzzy neural network (FNN) and hybrid genetic-based SVR (HGA-SVR). The outcome of their experiment demonstrated that HGA-SVR is the best algorithm and the feng shui model has a better performance in BPN, FNN and HGA-SVR [27]. Wu et al combined fuzzy reasoning technique with neural network to get an algorithm that has the ability in fuzzy reasoning and learning. According to the researcher, the combination gave a better and accurate prediction of the real estate prices than traditional neural network [43]. Dongmei combined genetic algorithm (GA) and BP neural network and compared the outcome of his experiment with the traditional methods [5]. The experiment demonstrated the importance and relevance of an early-warning system as a necessary and vital condition for a sustainable and stable development of buying and selling of properties in China. The experiment suggested that early-warning indicators should be improved as well as the predictive capability of the real estate market [48]. Ahmed developed a neural network model to predict the housing prices in the United States. The goal of the study was to have an algorithm that could be useful to real estate investors to make appropriate, timely and prudent financial decisions in regard to investing in real estate. He used historical market data sets as the training data; the experiment produced an error in the range of -2% to +2% [20].

Wedyawati and Lu [39] designed a data warehouse consisting of real estate properties listings. The experiment was performed to help prospective buyers and sellers of real estate to determine property prices. Oracle data warehousing tool kits were used in the construction of the data warehousing [11]. Data from Multiple Listing Services (MLS) were extracted into data warehouse and the researcher designed a star schema with a large fact table and three dimensional tables. With this set up, the researcher was able to use a linear regression model to predict the value of houses in the United States. Wu et al. proposed a multi-layer feed-forward neural network to determine the market value of real estate properties. To solve the problem of 'local maximum' associated with the use of the algorithm, the researcher suggested the use of Levenberg-Marquardt algorithm. The neural network model used for the study had 3 neurons in the input layer, 4 neurons in the hidden layer, while the output layer had 1 neuron [44].

John [18] researched into the dynamic effects of four common key macroeconomic variables using a nonstructural estimation on real estate prices and the stock of houses sold at a particular time. The outcome of his experiment suggested that employment growth and mortgage rates strongly affects the real estate market. Lu et al [28] have demonstrated the impact of real estate prices on resident income in China. The rising price of properties is a strong factor in increased polarization between the rich and the poor. Poor homeowners are forced to live in a particular part of the city while the affluent and rich homeowners are concentrated in other parts. The study showed that the map of China has been indirectly influenced by income polarization [10, 32].

The housing market operates in different forms across urban areas in China. A suggested good approach to analyze this trend is to apply an improved stock-flow model. The model according to the researchers is capable of identifying the relationship between different factors affecting demand and supply in the housing sector [13, 23]. The researchers performed an experiment to show that a generalized least square equation can be used to identify factors affecting housing prices for existing houses and new construction. The outcome of the experiment suggested that household income is the strongest factor affecting housing prices in China. However, another researcher in China demonstrated the impact of government taxes in controlling who owns a home. According to the study, government taxes are the strongest factor that affects the equilibrium price of real estate properties. The study showed that at 0.4% taxes, prices of real estate properties dropped by 8% and when the rate was increased to 0.9% prices dropped by 20% [34]. The outcome of the experiment suggested that while household income may affect prices of properties, taxes imposed by the government and local authorities have a greater impact on who owns a house in China. An uncontrolled price may result into inflation which may affect the standard of living of the Chinese population [41].

2.2 Linear Regression

A linear regression algorithm has the capability of estimating, evaluating and predicting a continuous target variable. The algorithm is based on the statistical modeling of the relationship that exists between an explanatory variable X and a response variable Y. The goal of a linear regression is to find the best straight line that fits a given dataset. In the simplest form, X could be a univariate while in a more complex expression, X could be multivariate. In other words, a linear regression generates an algebraic formula to estimate or predict where an observation from a testing dataset falls along an imaginary straight line through the training data; each explanatory variable X is evaluated statistically to predict response variable Y [9, 38]. In predicting the value of the response variable Y, there seems to be an associated predictive error. A linear regression error reduction is the square difference between the value of the hypothesis and the actual value of Y; this is normally referred to as squared error cost function. We can minimize the cost function using gradient descent. Gradient descent is based on using calculus to estimate the partial derivative of the cost function at each iteration to produce a local minimum [10].

2.3 Artificial Neural Network

Artificial neural network has been very useful in analyzing large data generated as a result of the extensive use of internet applications by millions of people all over the world. The learning algorithm was built to resemble the biological neuron systems of a human brain. Resembling a human brain, an artificial neural network consists of a complex network of nodes linked together to process information. Therefore, with a proper use of ANN, scientists have been able to get a machine that can process information and mimic the human brain. The simplest form of artificial neural network is a perceptron which is widely used to solve classification problems. A perceptron calculates weighted sum of its inputs x and a bias factor z to get an output y [1]. Unlike a perceptron, a more complex artificial neural network consists of several nodes, multiple input features and activation functions. In between the output and input layers is the hidden layer. The hidden layer may be multiple layers of nodes.

2.4 K-NN

The k-NN rule classifies unknown example *x* based on its closest or nearest neighbor. In a multidimensional space, a testing example x, is classified on the class of its k nearest neighbor [17]. In other words, given an unknown test example *x*, the algorithm searches the multidimensional space to find the pattern k nearest neighbors that are closest to *x*. For n attributes, each attribute is represented by a data point, thus a distance measured of points closest to each testing example is computed. Neighbors are determined from the training examples of which correct classification has been known or in case of regression, labelled. K is a predetermined number of closest or nearest neighbors of the unknown example *x* [12].

2.5 Polynomial Regression.

A linear regression assumes that the relationship between the explanatory variable and the response variable are always linear. However, there are cases where the relationship is non-linear. In this particular situation, using linear regression produces misleading results. Polynomial regression has the capability of a higher performance in an n-dimensional space. A linear regression equation is easily transformed to a polynomial regression by using coefficient terms with a higher degree. This transformation makes it possible for a polynomial regression learning algorithm to fit into a curvilinear relationship between the explanatory and response variables.

3 Predictive Software Application (Phase One)

3.1 Real Estate Software Modeling

We designed a real estate data warehousing star schema and implemented it using Oracle 11g. Data warehousing was used for the experiment because it is easy to understand, analyze and information retrieval is less cumbersome [1]. The schema comprised of a large fact table with three dimensional tables. All relevant data from MLS were loaded into the table. There are four main steps in building a data warehousing: extraction, transformation, modeling and transport [39]. We extracted our data from Multiple Listings Services (MLS). MLS is the main database of real estate properties in the United States. In most cases, each metropolitan area has its own MLS. The MLS provides a medium where buyers and sellers of real estate properties meet through their agents. The data we used for the experiment was taken from the Washington DC metropolitan area of the United States. Data obtained were in a flat form format and stored in a CSV format.

We used Oracle 11g Data warehousing to build a warehouse called Howard. After extraction, our data went through transformation stages. In the MLS, each property has status; active, off market, expired, pending, contract, contract with kick out, contingent and withdrawn. Part of the transformation process was to identify all error data, detect duplicates and separate all relevant data for processing. Data obtained from the MLS were in CSV, we transformed it to a ‘dat’ file for processing on Oracle. The ‘dat’ file was loaded to the warehouse using SQL* Loader utility. Data were loaded to a table called Howard table, from where we created a large fact table called Real Estate Fact Table. We created three additional tables; School, Listing Agent and Listing Broker dimensional tables to store information about school, listing agents and listing brokers respectively. Scripting and connection were done using PHP and Apache respectively. PHP and Apache were suitable because of their flexibility and compatibility with the Oracle 11g. The software created offers real estate stakeholders – buyers, sellers, property appraisals, investors and financial institutions an interactive portal where they can list and search properties. Unlike the Multiple Listing Services (MLS), our software is accessible to all major stakeholders in the real estate business with an added advantage of verifying values of properties. In other words, real estate transactions can take place without real estate agents. (Sellers have the option of listing properties without going through listing brokers, while buyers and banks can verify values without requesting for an appraisal).

We used forward engineering in the design and development of PREMLS; created code from the model. A considerable effort was put into the design in ensuring that a good model was in place before we began the implementation. UML modeling was used in the design and development of the software; this was crucial in determining all necessary objects. The static structure (behavior) of the system was represented using class diagrams to show the class, multiplicity and association. We considered modeling to consist of developing and building the abstraction of the system. Use cases were used to determine the elicitation requirement of PREMLS. Use case was considered as a form of abstraction used to describe a class of scenarios in building a software. Each use case was documented with a textual description. A complete textual use case consists of the name of the use case, participating actors, entry condition, flow of events, exit conditions and special condition (if necessary). We used ten different use case scenarios for the system. Scenarios describe the use of the system as a series of interactions between different stakeholders of real estate transactions and PREMLS

Abbott’s technique was used to generate participating objects in each of the use cases that were described. Abbott’s technique is based on the principle of producing a textual analysis of a textual use case based on noun-verb occurrences in the requirement elicitation. We considered nouns as candidates for objects/classes, while verbs were considered as candidates for operations. We used use case diagrams to describe the functional behavior of the system. Use case diagram is a diagrammatic expression of the use of the system as seen from the perspective of different stakeholders. Figure 1 shows one of the ten use case diagrams used in developing PREMLS. As shown in the diagram, actors are home-buyer, property appraisal, home-seller and real estate agent (realtor). Each actor has different levels of interaction with the system.

PREMLS design was based on the nature of real estate business in the United States. Non-functional behaviors considered in building the system include: the capability of the system to maintain functionality even if the user enters a wrong input (robustness), the ease with which all stakeholders can use the system to perform a function (usability) and the ratio of the expected uptime of the system to the total sum of the expected up and down time (availability).

The system architectural design was based on model-view-controller (MVC) architectural style; we considered this as the solution domain. Figure 2 shows the architectural design of the system. The model-view-controller architectural style is the preferred architecture for our system because it has the advantage of eliminating coupling between the model and boundary objects. This is very important because changes in the user interface do not disrupt the knowledge domain. Also MVC enhances efficiency by increasing coherency among classes/objects. We considered coherency as a measure of interdependencies among classes. Added to these advantages is the fact that MVC is a triangular architectural style [4].

3.2 Data Collection

Using the graphical user interface, we retrieved 135 houses from the database of all houses that were sold between January 1 and March 15, 2015. Datasets were pruned to only properties with sold status. We used sold properties because in the real estate world, properties with status sold must have passed property appraisal and inspection. This is necessary because

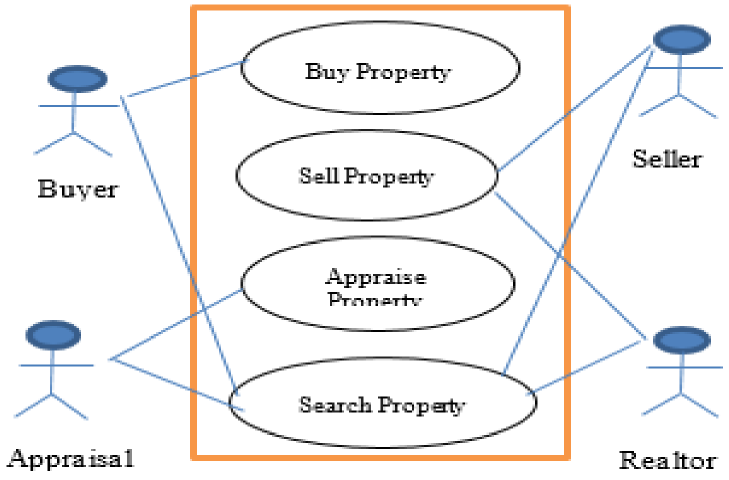


Figure 1: A use case diagram

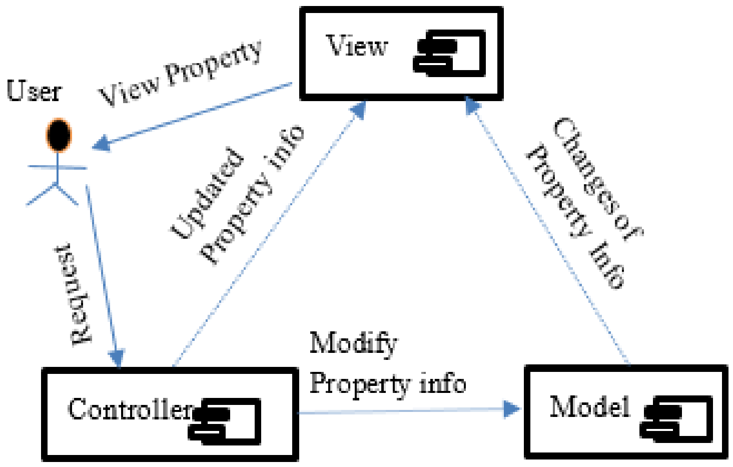


Figure 2: MVC architectural design of PREMLS.

property showing active listing status or any other status may not pass property appraisal or inspection, therefore, they do not represent the true price of the property. Property appraisal is the fair market value of a property as determined by a government licensed property value estimators called property appraisers. Licensed appraisers based their opinion on the comparison of at least three recently sold properties of the same features closest to the listed property in the recent time. Basically, licensed appraisers use the average price of other properties to estimate the value of a given property. Appraisal report often gives little weight to interior improvement (properties value is a function of surrounding properties); thus, it is realistic to design a value predictor that will be close to the appraised value of a property. Licensed home inspectors on the other hand inspect properties to document the physical condition of the property prior to a sale. Property that fails home inspection may disqualify a sale or reduces the fair market value of the property. Each of the properties that we retrieved from PREMLS has a reference number, closed priced, number of full baths, number of half baths, number of levels, number of square footage, year built and number of fireplaces.

3.3 Learning Algorithm

We used linear regression as the predictive learning algorithm. A linear regression algorithm maps a variable *Y* (predictive variable) to a given input variable *X* (predictor variable) [49]. The algorithm is suitable for our experiment because of the small sample size of the datasets. The linear regression equation is: $Y = \alpha + \beta X$, where, *Y* is the fair market price of the property, *X* represents other variables (bedrooms, half baths, full baths, square footage, fireplace, level and year built), α and β are the intercept term and regression coefficients respectively. We used a multivariate linear regression for the experiment. While a single feature may be a good indication of the price of a property; however, it is not sufficient to produce a fair market value. Therefore, we preferred an algorithm that uses more than one single feature. Mathematically, a multiple linear regression [10] is represented as:

$$y = \alpha + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_n x_n \quad (1)$$

The formula shows that each explanatory variable, *x*, of our housing feature has an arbitrary coefficient β. Our goal is to get a combination of the weights β and the intercept term α that generates the least error for the *y* value.

Now,

$$Y = X\beta \quad (2)$$

Therefore,

$$\beta = \left(X^T X\right)^{-1} X^T Y \quad (3)$$

For programming purposes, the matrix of the expression is represented as:

$$\begin{bmatrix} Y_1 \\ Y_2 \\ \vdots \\ Y_n \end{bmatrix} = \begin{bmatrix} \alpha & + & \beta X_1 \\ \alpha & + & \beta X_2 \\ & \vdots & \\ \alpha & + & \beta X_n \end{bmatrix} = \begin{bmatrix} 1 & X_1 \\ 1 & X_2 \\ \vdots & \vdots \\ 1 & X_n \end{bmatrix} \times \begin{bmatrix} \alpha \\ \beta \end{bmatrix} \quad (4)$$

Where *Y* is a column vector of the price of properties (Predictor), β is the column vector of the coefficient of the housing data parameter. X represents the explanatory variables in the datasets (m by n dimensional matrix). The number of our training example is m, while n is the number of explanatory variables.

RapidMiner and Scikit-learn (a python programming library) were used for the experiment. We used RapidMiner to build a prototype and examine the datasets, while Scikit-learn was used for the analysis. Building a prototype was necessary so as to know the viability of the experiment, the spread of the data and some basic statistical summary of the datasets. The statistical summary showed that features like number of bedrooms have a strong relationship with the price of a property, while the number of levels has a weak relationship. Therefore, we concluded that the price of a property depends on a multiple of explanatory variables. Datasets were loaded into Scikit-learn using pd.read_csv() function. Datasets were divided into two; training and testing sets; 25% of the data was set to test while 75% was set to train. The reference property number was used as property identifier, prices as the response variable and remaining features were used as the explanatory variables.

We trained the model and evaluated its outcomes on the test data. Gradient descent was used to find the best fit for the algorithm.

4 Result and Analysis

As proposed, the predictive model was designed using linear regression. Linear regression as explained has the capability of generating α and β, the intercept term and regression coefficients respectively. The intercept term is the point where the x axis (explanatory variable) crosses the y axis (response variable). In predicting the value of a property, using (1, 2, 3 and 4) we obtained the combination of these terms that best fit our model. Table 1 shows the weight (co-efficient parameter) of each explanatory variable and the intercept term. Each feature of the explanatory variable produced a correspondent weight (coefficient parameter). We plotted the graph of the prediction versus the target; estimated price of properties versus the appraised price (closed price) of the property as listed on the MLS. Using our testing datasets, the graph shows that most values produced by our algorithm matched the actual value. Figure 3 shows a graph comparing our prediction with the target values. Target values are the actual values of properties on the Multiple Listing Services (MLS).

Table 1: Weight of each Explanatory Variable

Feature	**Weight**
Bedroom	3.79975315e+04
Baths Full	4.23472313e+04
Baths Half	2.93415223e+04
Levels	-4.19315566e+04
Fireplaces	6.17806409e+04
Lot square footage	2.34963310e-01
Year Built	2.51348690e+03
Intercept Term	-4.70102955e+06

Performance evaluation was done using r-squared score. R-squared is a performance evaluation value that shows how well the model fit the target value. We used a combination of residual sum of squares, mean-squared error, and root mean-squared error functions to obtain the value of r-squared. R-squared is suitable because it shows the accuracy of the testing variables to the prediction of the model. An r-squared of one indicates that the error in predicting a testing variable is zero, however, this is not very common in experiments. An r-squared that is more than a half shows that a greater part of the variance is explained by the model. Thus, r-squared takes a value between zero and one, in rare cases it generates a negative number if the model's performance is extremely poor [4].

5 Implementation of Real Estate Fair Market Value Predictor

We integrated the outcome of our experiment into our multiple listing software and called it Predictive Multiple Listing System. Unlike the Multiple Listing Service, a user of PREMLS has the option of estimating the value of a property by simply entering the features of the property on the 'predict' portal of the system. An interactive portal was created for users to input the features of a property (number of bedroom, full baths, half baths, levels, fireplaces and square footage etc.). The sequence diagram of viewing and predicting a fair market value of a real estate property is shown in Figure 4. The sequence diagram shows the dynamic behavior of PREMLS. We represented classes by columns, arrows represent messages, and the narrow rectangles represent activations. The source of an arrow shows the activation that sends a message and lifelines are indicated with the vertical dashed lines [33]. The figure shows the information flow of how a home buyer or any other actor can use the system to view properties and predict fair market values:

1. The home buyer triggers the ViewProperty method in an instance P of the PropertyInfo object class.
2. P sent a message to the system.
3. User authorization was checked.
4. If authorized, information of properties within a price range are returned.
5. User triggers PredictProperty().
6. Fair market value returned.
7. If authorization failed, an error message is generated.

6 Comparison Study (Phase Two)

6.1 Problem Description

The second phase of our experiment investigated various

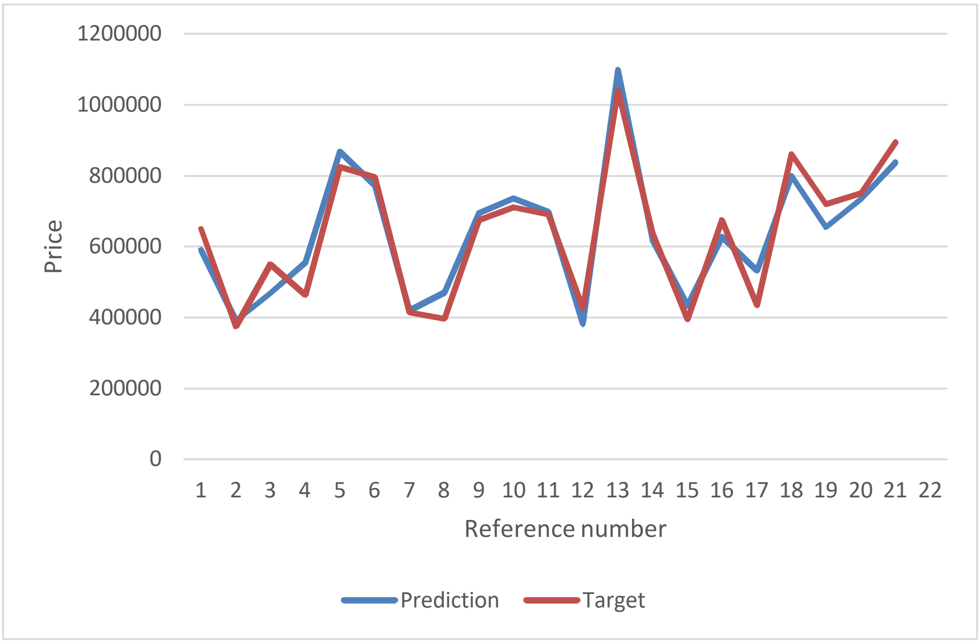


Figure 3: Comparison graph of price prediction versus appraised values

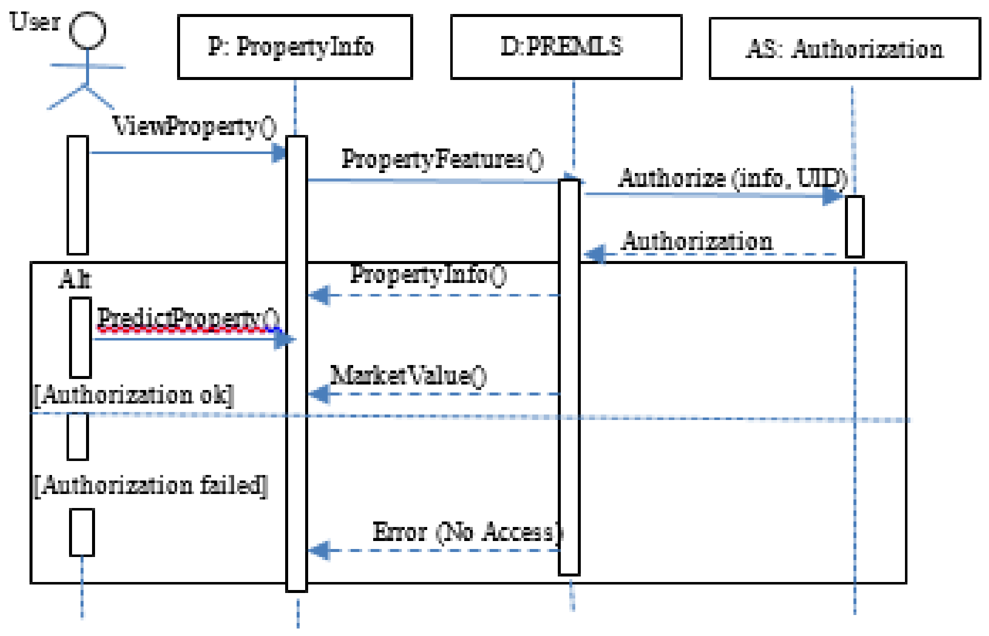


Figure 4: Sequence diagram of predicting a real estate value

factors that affect the predictability of the fair market values of properties. We used statistical inferences and learning algorithms as tools for the study.

6.2 Datasets

Six hundred and eighteen standard sales single family properties that were listed and sold between January and March 2015 in Montgomery and Baltimore Counties in the state of Maryland, United States were extracted, processed and analyzed. Datasets were collected from the Multiple Listings Services [15]. Information collected from the MLS are reliable, because it is the official repository of all real estate transactions in most metropolitan areas of the United States. Also, we built our model based on the parameters that real estate agents and appraisers use for a CMA (comparative market analysis) on the MLS.

6.3 Investigation Setup

Our hypothesis was based on the following:

1. The fair market value of a property is predictable using the datasets of properties listed on the MLS in a county.
2. The selection of learning algorithms has a considerable impact on the predictability of the fair market value of residential properties in the United States.
3. The statistical significance of features in different counties are not necessarily the same.

Statistical analysis was based on scatter plot, correlation matrix, covariance matrix and test of significance. Neural network, linear regression, K-NN and Polynomial regression were used as a learning algorithm to determine whether to reject or fail to reject the hypothesis. As shown in Figure 5, our dataset went through a pre-processing stage to make the data suitable for processing. The pre-processing stage include; removal of extraneous information, normalization, outliers and missing data detection. We used cross validation to partition the datasets into *k* subsets of equal sizes. One k subset was used as the testing dataset while the remaining *k -1* were used for training. Cross validation of the dataset was done *k* times. In each validation k subsets of the dataset was used only once as the training dataset. We used 10 as the value of *k*.

As indicated above, our learning algorithm comprises of linear regression, neural network, polynomial regression and KNN. For the linear regression model, we used the ridge regression model with Akaike criterion. Akaike criterion relatively

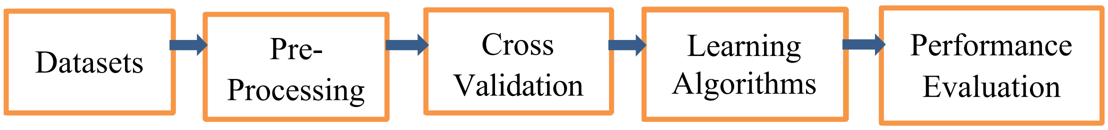


Figure 5: Block diagram of the predictive model

measures the fitness of the model. We used 500 training circles for the feed forward back propagation neural network algorithm and 0.3 as the learning rate. Error epsilon was set to 1.0E-5 - optimization of the model stopped if the training error gets below this threshold. To avoid over fitting, we used a maximum degree of 5, minimum coefficient of -100 and a maximum coefficient of 100 for the polynomial regression model. For the K-NN model, we set the value of *k* to 1 so that the testing datasets were easily assigned to its nearest neighbor. Mixed Euclidean distance was used to measure the relative distance between data points. Performance of each model was measured using Spearman's rho. Spearman's rho shows the correlation between the actual value of the property and its predicted price.

6.4 Experimental Results and Analysis

The purpose of analyzing the experimental result was to accomplish the following:

1. Confirm (or fail to confirm) the parameters that actually influence the fair market value of a property.
2. Verify the presumed linear relationship between the explanatory variables and the response variable (price of a property).
3. Determine the co-efficient values of the explanatory variables that best fit the relationship.
4. Compare the statistical significant value of each parameter in the two counties.
5. Outline the efficiency of different learning algorithms using Spearman's rho.
6. Compared the Spearman's rho value of each learning algorithm.

6.4.1 Statistical Analysis. As stated above the same sample size of the population of single residential properties listed on the MLS in Montgomery and Baltimore counties, USA, between January and March 2015 were used for the experiment. We considered the correctness and completeness of datasets as a necessary measure to reduce error of inductive inferences. One important research question is "is there a statistical relationship between the value of a property and the explanatory variables considered for the study?" If the answer to this question is yes, then the next question is "are the relationships the same in all counties of the United States?" We considered the two questions using scatter plot, covariance matrix, correlation matrix and tests of significant of the sample data sets of the two counties in the study.

6.4.1.1 Scatter Plot. Scatter plots were used to verify associations between the parameters. A scatter plot reveals if two variables are linearly or non-linearly related. Outliers are also clearly identified in scatter plots. Figure 6 shows the scatter plot of the Prices vs Latitude in Baltimore County. The figure shows that the relationship between the parameters exhibits a non-linear shape. A non-linear relationship is an indication that changes in latitude does not correspond to a change in the price of property in Baltimore County.

Figure 7 shows the scatter plot of the prices of properties versus bedrooms in Baltimore County. The figure shows that while some data are outliers, a large number shows a positive relationship between prices of properties and number of bedrooms. Thus, increasing the number of bedrooms may increase the market value of a property.

6.4.1.2 Covariance Matrix. Statistical association between the explanatory and response variables in the two counties were also examined using a covariance matrix. A covariance matrix takes the difference of two parameters from their means and multiplies them together. If the product is positive, the two parameters are said to have a positive covariance, otherwise a negative covariance or a zero covariance. Positive covariance indicates that the two parameters vary in the same direction, while a negative covariance shows the contrary. For example, in the Montgomery, our analysis shows that the covariance of bedrooms and price has a value of 135130.639, it means that both parameters vary in a positive way. The analysis also shows that there is a -4698.777 covariance between price and longitude in the same county. A negative covariance is an indication that the two parameters exhibit an opposite trajectory. It should be noted however, that since both parameters are not in the same scale, a high positive value does not necessarily mean a strong

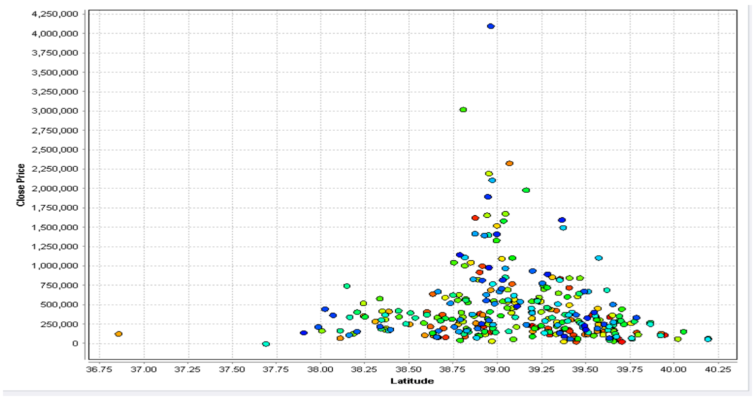


Figure 6: Baltimore close prices vs latitude

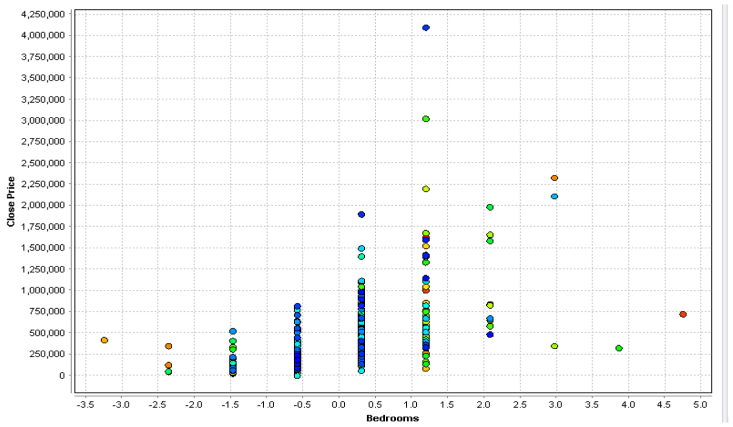


Figure 7: Scatter plot of the prices of properties versus bedrooms in Baltimore County

covariance relation, it just is an indication of association between parameters. Covariance is an early indication of the relationship between two attributes. A further test is necessary to validate the strength of the relationships.

6.4.1.3 Correlation Matrix. We further examined the datasets and obtained a correlations matrix between the parameters. Correlation is a measure of the strength or degree of statistical relationship that exist between two variables [30]. Since attributes were of different sizes it was necessary to rescale them to the same range, therefore, normalization was done as one of the preprocessing stages. Datasets were normalized into a normal distribution with mean set to zero and variance to 1. A correlation matrix of the variables were obtained. Table 3 shows the correlation matrix of Baltimore and Montgomery counties. The table shows that the parameter in each county have different degrees of association or relationships. The matrix shows an 11 by 11 matrix because 11 features were considered for the study. Values in each cell ranges between -1and 1. On the diagonal axis, each cell is 1 because at this point a feature is mapped to itself, while values at other cells are less than 1. A positive number indicates that there is a positive correlation between the variable, on the other hand a negative number shows that there is a negative relationship. A high positive number is an indication that there is a strong relationship between the two entities, which may be

Table 3: Correlation matrix of parameters

Pa-rameter	County	Lot Sqft	Bedrooms	Baths Full	Baths Half	Levels	Fireplaces	Basement	Year Built	Close Price	Latitude	Longitude
Lot Sqft	Montg.	1	0.105	0.200	0.035	-0.053	0.251	-0.014	0.107	0.140	0.219	-0.267
	Baltimore.	1	0.056	0.716	-0.109	0.051	0.062	0.018	-0.266	0.048	0.016	-0.067
Bed-rooms	Montg.	0.105	1	0.620	0.190	0.257	0.230	0.142	0.413	0.443	0.060	-0.163
	Baltimore.	0.056	1	0.175	0.361	0.475	0.355	0.322	0.128	0.517	-0.014	0.148
Baths Full	Montg.	0.200	0.620	1	0.066	0.182	0.373	0.155	0.403	0.615	-0.071	-0.230
	Baltimore	0.716	0.175	1	0.017	0.131	0.107	0.100	-0.114	0.173	-0.012	-0.08
Bath Half	Montg.	0.035	0.190	0.066	1	0.397	0.219	0.165	0.381	0.273	0.057	-0.035
	Baltimore.	-0.109	0.361	0.017	1	0.397	0.360	0.281	0.183	0.441	0.133	0.094
Levels	Montg.	-0.053	0.257	0.182	0.397	1	0.117	0.414	0.172	0.285	-0.104	0.010
	Baltimore.	0.051	0.475	0.131	0.397	1	0.329	0.643	-0.035	0.382	0.227	0.020
Fire-places	Montg.	0.251	0.230	0.373	0.219	0.117	1	0.174	0.160	0.455	-0.080	-0.126
	Baltimore.	0.062	0.355	0.107	0.360	0.329	1	0.211	-0.144	0.556	-0.020	0.012
Base-ment	Montg.	-0.014	0.142	0.155	0.165	0.414	0.174	1	0.110	0.160	-0.148	0.070
	Baltimore.	0.018	0.322	0.100	0.281	0.643	0.211	1	0.142	0.222	0.214	-0.026
Year Built	Montg.	0.107	0.413	0.403	0.381	0.172	0.160	0.110	1	0.225	0.406	-0.344
	Baltimore.	-0.266	0.128	-0.114	0.183	-0.035	-0.144	0.142	1	0.138	-0.180	0.123
Close price	Montg.	0.140	0.443	0.615	0.273	0.285	0.455	0.160	0.225	1	-0.440	-0.151
	Baltimore.	0.048	0.517	0.173	0.441	0.382	0.553	0.222	0.138	1	-0.079	0.123
Lati-tude	Montg.	0.219	0.060	-0.071	0.057	-0.104	-0.080	-0.148	0.406	-0.440	1	-0.432
	Baltimore.	0.016	-0014	-0.012	0.133	0.227	-0.020	0.214	-0.180	-0.079	1	-0.118
Longi-tude	Montg.	-0.267	-0.163	-0.230	-0.035	0.010	-0.126	0.070	-0.344	-0.151	-0.432	1
	Baltimore	-0.067	0.148	-0.080	0.094	0.020	0.012	-0.026	0.123	0.123	-0.118	1

interpreted as a strong correlation; both attributes fall and rise together. A negative number shows that the increase of one attribute produces a decrease in the other attributes (movement is in opposite trajectories). While more experiment is still necessary, however the correlation matrix is another indication of the strength of association between the parameters, a further test is necessary to validate the relationships.

6.4.2 Test of Significance and Linear Regression. Test for significance was used to confirm the relationship between the variables. The research question here is "what is the probability that the assumed statistical relationships between the explanatory parameters and the fair market values of properties are not random chance?" The question was answered by conducting a t-test for the study. We selected a p-value of 0.05 to conduct the test. A p-value is a probability value that shows a considerable evidence on whether to reject the null hypothesis. The null hypothesis in our study stated that a given explanatory feature does not have effect on the value of a property. A p-value that is less than 0.05 on a "one-tailed" t-test is considered significant evidence against the null hypothesis. In other words, changes in the explanatory feature value are strongly related to changes in the value of a property. Conversely, a high p-value (p>0.05) indicates that changes in the explanatory feature value are not related to changes in the value of a property. Table 4 shows the linear regression and test of significance tables for Montgomery and Baltimore counties respectively.

Considering the test of significance of both counties, the statistics shows that while some features are statistically significant in some counties, they are not significant in other counties. For example, a high p-value of longitude and latitude in Baltimore shows that the two features are not statistically significant in designing an accurate model to predict the price of a property. The table shows that each explanatory variable has different coefficient values in both counties.

The code column indicates the degree of significance of each parameter (one star represents minimum and four stars represents maximum). Coefficient values indicate that a unit change in a given property parameter, produces an estimated change in the value of a property, all other parameters being constant. For example, in Montgomery County, for every additional increase in the number of bedrooms (independent variable), the value of a property (dependent variable) increases by an estimated value of 41329.940, given that all other independent variables are held constant.

Since the parameters are on different scales, the Standard Coefficient column shows the comparison of the weights on a uniform scale. It reveals which parameter has the highest influence in determining the value of a property. The table shows that in Montgomery County, prices of properties are mostly influenced by the number of full baths with a standardized coefficient of 0.315. On the other hand, the table also shows that fireplaces with a standardized weight of 0.392 has the greatest influence on the value of a property in Baltimore County.

6.4.3 Neural Network. The neural network used for the experiment had one input layer, one hidden layer and one output layer. The input layer had twelve nodes (each node represents one parameter with an additional node for bias), the hidden layer had seven nodes and a bias node, while the output layer had only one node. The output of the input layer served as the input to the hidden layer (nodes in the input layer contributed a weighted sum to each node of the hidden layer). For example, in Montgomery County, the bedroom node contributed 0.692 weighted input to node 1of the hidden layer, while basement node contributed -0.432 of its weighted value to node 5. In

Table 4: Test of significance and linear regression

Parameter	County	Coeff.	Std. Error	Std. Coeff.	Tolerance	t-start	p-Value	Code
Lot Sqft	Mont.	27139.550	12239.213	0.076	0.991	2.217	0.031	**
	Balt.	N/A	N/A	N/A	N/A	N/A	N/A	
Bedrooms	Mont.	41329.940	14899.389	0.116	0.794	2.774	0.006	***
	Balt.	112767.376	23654.386	0.239	0.715	4.767	0.000	****
Baths Full	Mont.	112414.466	16503.883	0.315	0.661	6.811	0	****
	Balt.	45287.392	19807.205	0.096	0.986	2.286	0.026	**
Bath Half	Mont.	55765.308	13596.485	0.156	0.959	4.101	0.000	****
	Balt.	75152.207	22940.589	0.159	0.771	3.276	0.001	***
Levels	Mont.	25847.826	13527.424	0.073	0.929	1.911	0.068	*
	Balt.	64860.334	28663.123	0.138	0.788	2.263	0.028	**
Fireplaces	Mont.	60559.255	12759.997	0.170	0.844	4.746	0.000	****
	Balt.	185112.690	22370.044	0.393	0.814	8.275	0	****
Basement	Mont.	-25683.903	12538.227	-0.072	0.932	-2.048	0.048	**
	Balt.	-39941.929	26022.329	-0.085	0.852	-1.535	0.159	
Year Built	Mont.	39146.742	15586.309	0.110	0.959	2.512	0.014	**
	Balt.	68290.883	21633.187	0.145	0.999	3.157	0.002	***
Latitude	Mont.	-206675.413	14384.326	-0.580	0.982	-14.368	0	****
	Balt.	-3.2952.384	20761.872	-0.070	0.999	-1.587	0.142	
Longitude	Mont.	-78627.878	13129.799	-0.221	1.00	-5.989	0	****
	Balt.	21235.428	19748.557	0.045	0.987	1.075	0.412	
Intercept	Mont.	633958.266	1134.530			56.936	0	****
	Balt.	457279.324	19117.745			23.919	0	****

Baltimore County, the bedroom node contributed 0.064 weighted input to node 1of the hidden layer, while basement node contributed 0.213 of its weighted value to node 5. Table 5 shows the weight activation for the hidden unit of Montgomery and Baltimore counties.

The output of nodes in the hidden layer contributed a weighted sum into the final output layer. Errors were propagated backwards in the network. The table shows that the weights of each of the parameters are different in both counties.

6.4.4 Polynomial Regression. The polynomial regression unlike a linear regression has a degree that is equal to or greater than one in the coefficient value. While linear regression is capable of modeling linear relationships between two variables, a polynomial regression can model a non-linear relationship. Table 6 shows the combinations of α and β, the intercept term and regression coefficients respectively in each county with the exponential terms that best fit the model.

Each explanatory variable in each county produced a corresponding weight with their exponential terms. The intercept term is the point where the explanatory variable crosses the response variable (fair market value). The table shows that the weights of each of the parameters are different in both counties.

6.5 Performance Comparison

We compared the performance of each learning algorithm in Baltimore and Montgomery counties using Spearman's rho. Spearman's rho is a monotonic functional description of the statistical relationship, association or strength of linkage that exists between two parameters. Spearman's rho is suitable to compare the performance of each learning algorithm because it shows the degree of association between the target variable (price of a property as predicted by appraisers) and our predicted value. A Spearman's rho value is between -1 and 1. Experimental outcome with a positive Spearman's rho value is an indication that the predicted value and its targeted variable has a positive correlation, while a negative value shows a negative relationship. In other words, a positive Spearman's correlation coefficient shows that both variables (target and predicted) increases and decreases in the same direction. A higher positive value means a strong correlation, while a high negative value shows the two variables trajectories are in extreme opposite. Spearman's rho value of zero shows that there is no statistical relationship between the variables. We obtained the Spearman's rho of each learning algorithms in Baltimore and Montgomery counties. Performances were tabulated and plotted on a bar chart.

The outcome of our experiment shows that the predicted outcome of most of the learning algorithms have a strong association with the appraised price (actual price) of a property - our target value. Table 7 shows a comparison table of the learning algorithms. In the table, a 0.826 Spearman's rho value in the linear regression model of Montgomery County shows that the appraised (actual price) and predicted values are 82.6 % closer to becoming a perfect monotonic function of each other. On the other hand, a 0.266 performance of the polynomial regression in the same county is an indication that the predicted and the appraised values are 26.6 % closer to becoming a perfect monotonic function of each other. The table also shows that the performance of the four learning algorithms are better in Montgomery County than Baltimore County. Polynomial regression has the worst performance, while neural network has the best performance in both counties. Figure 8 shows the bar

Table 5: Output of the neural network input layer

Parameter	County	Node 1	Node 2	Node 3	Node 4	Node 5	Node 6	Node 7
Bedrooms	Montgomery	0.692	-0.745	1.017	-0.395	0.736	0.132	2.929
	Baltimore.	0.064	-1.532	0.789	-0.840	0.089	0.112	0.556
Full Bath	Montgomery	0.415	-0.605	2.655	2.121	0.355	0.404	-0.45
	Baltimore.	0.666	-6.539	0.777	1.492	0.652	0.601	2.151
Baths Half	Montgomery	0.438	-0.019	0.515	1.281	0.415	1.056	1.348
	Baltimore.	0.957	-1.439	1.076	1.342	1.018	0.112	1.129
Levels	Montgomery	0.365	-0.51	-1.434	-0.948	0.304	-0.80	-1.255
	Baltimore.	1.031	-1.583	1.196	0.081	1.087	0.885	0.175
Fireplaces	Montgomery	0.921	-0.094	-0.425	1.103	0.810	0.571	-1.589
	Baltimore.	0.137	-2.624	0.490	0.762	0.176	0.115	-3.366
Basement	Montgomery	-0.323	-0.056	-0.545	-0.112	-0.432	0.428	0.142
	Baltimore.	0.308	-0.448	-1.019	-0.070	0.213	0.337	0.119
Lot Sqft	Montgomery	1.433	-2.113	0.263	2.103	1.259	1.097	2.722
	Baltimore.	0.585	-0.317	-0.296	0.364	0.523	0.710	0.498
Year Built	Montgomery	-0.605	1.199	2.294	0.600	-0.347	1.058	-0.316
	Baltimore.	-1.332	-7.932	-1.325	-1.008	1.339	1.156	0.213
Latitude	Montgomery	-0.182	2.498	-0.258	-0.020	-0.115	-0.645	-1.421
	Baltimore.	0.958	0.314	-1.073	2.485	-0.993	-0.828	-1.868
Longitude	Montgomery	-0.417	1.038	-0.583	-2.503	-0.440	-0.640	0.559
	Baltimore.	-0.058	-0.263	-0.293	-0.778	-0.059	-0.111	-0.279
Bias	Montgomery	-1.093	0.144	-3.905	-1.138	-1.210	-1.676	-2.656
	Baltimore.	-0.568	0.608	0.304	-1.026	0.625	0.559	-2.148

Table 6: Polynomial regression table

Feature	**County**	**Weight**	**Exp**
Bedrooms	Montgomery	53.514	1
	Baltimore	- 40.640	1
Baths Full	Montgomery	100	5
	Baltimore	68.919	3
Baths Half	Montgomery	55.359	3
	Baltimore	- 44.177	1
Levels	Montgomery	99.995	2
	Baltimore	100	5
Fireplaces	Montgomery	100	5
	Baltimore	51.095	5
Basement	Montgomery	89.473	2
	Baltimore	100	2
Lot Sqft	Montgomery	100	4
	Baltimore	99.996	3
Year Built	Montgomery	-75.722	1
	Baltimore	-71.153	1
Latitude	Montgomery	55.613	5
	Baltimore	93.044	4
Longitude	Montgomery	-35.30	5
	Baltimore	- 19.768	3
Intercept Term	Montgomery	37.318	
	Baltimore	61.057	

chart.

7 Future Work

The outcome of the experiment showed that the design of PREMLS is successful at predicting the fair market value of real estate properties. However, our future work includes enlarging the algorithm to be applicable to condominium properties where factors such as condominium association fees factored into the price of a real estate unit. Also, factors such as interest rates, government policies, fraud, sentiments and media bias may have significant effect on the real estate market. Any of these factors or combinations of them may have a substantial effect on the algorithm.

8 Conclusion.

The first phase of our study examined the capability of a predictive real estate multiple listing system using MVC architecture with a linear regression. The experiment showed that using multivariate explanatory variables, we predicted the fair market value of real estate properties values using linear regression. Raw data from the Multiple Listing Services (MLS) of real estate agents from the Howard County, Maryland, USA, were used as training and testing datasets. The accuracy of the algorithm was measured using r-squared. R-squared was defined as an indication of how well the model fit the target value. The outcome of the study indicated an r-squared of 0.92. As a prerequisite to

Table 7: Performance comparison table

	Linear Regression	Neural Network	K-NN	Polynomial Regression
Montgomery County	0.826	0.836	0.699	0.266
Baltimore County	0.713	0.739	0.697	0.11

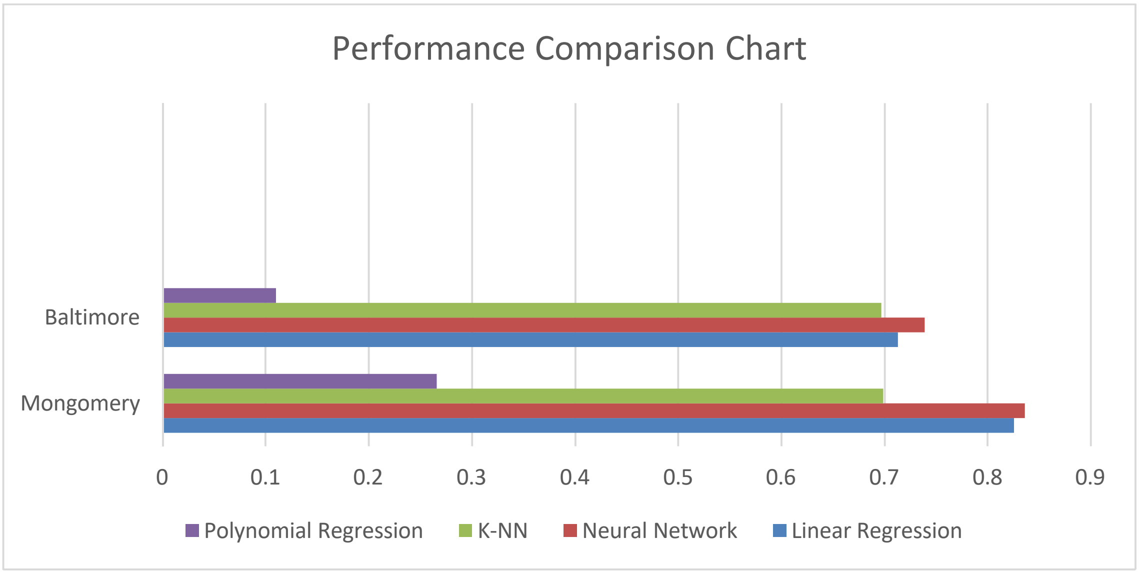


Figure 8: Learning algorithm performance comparison. Model performed differently in each county

the experiment, we developed a real estate software application using Oracle 11g warehousing as the knowledge domain. PHP and Apache were used to create a dynamic interface between the user and the knowledge domain. We used forward engineering in the design of PREMLS. Use case scenarios were used for the requirement elicitation, Abbot technique was used for analyzing textual use cases. Use case diagrams were used to determine the functionality of the system. The static structure (behavior) of the system was represented using class diagrams to show the class, multiplicity and association. Sequence diagram was used to illustrate the dynamic behavior of the system.

PREMLS has the capability of delivering a robust, usable, efficient and dependable real estate software to the public. Unlike the MLS, our software is accessible to all major stakeholders in real estate business with an added advantage of predicting values. In other words, using PREMLS, real estate transactions can take place without real estate agents. Sellers have the option of listing properties without going through listing brokers, while buyers and banks have the privilege of verifying estimated values without requests for an appraisal.

The second phase of our study investigated different factors affecting the predictability of the fair market value of a property in a county. We examined the performance of four different learning algorithms; neural network, K-NN, linear regression and polynomial regression. Real estate datasets of properties from Baltimore and Montgomery counties in the state of Maryland, United States were processed and analyzed. Performance comparison of learning algorithms was done using Spearman's rho correlation coefficient. Statistical analysis were conducted with scatter plot, correlation matrix, covariance matrix and test of significance. The outcome of our investigation shows that:

1. There exists a strong evidence of statistical relationship between the explanatory variables of datasets of properties listed on the Multiple Listing System and the fair market values of properties.
2. The fair market value of a property is predictable using datasets of properties listed on the Multiple Listing System.
3. The selection of learning algorithms has a considerable impact on the predictability of the fair market value of residential properties.
4. While some features in a real estate dataset are statistically significant in some counties, they are not significant in other counties.

Based on the statistical analysis of the result and the performance of the four learning algorithms in the two counties, the following are the implications of the study:

1. Different parameters influence the fair market value of properties in different counties.
2. The weighted values of each explanatory parameter in a predictive model are not the same in all counties.
3. Some learning algorithms are not suitable for predicting the fair market values of properties.

Acknowledgement

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

References

[1] Bruno Aiazzi, Stefano Baronti, and Luciano Alparone, "Lossless Image Compression Based on an Enhanced Fuzzy Regression Prediction," *Image Processing,* 1:435-439, 1999.

[2] Mohammad Abu Alsheikh, Shaowei Lin, Dusit Niyato, and Hwee-Pink Tan, "Machine Learning in Wireless Sensor Networks: Algorithms, Strategies, and Applications," *IEEE Communications Surveys & Tutorials*, 16(4):1996-2018, Fourthquarter, 2014.

[3] Angelos Amanatiadis, Loukas Bampis, and Antonois Gasteratos, "Accelerating Single-Image Super-Resolution Polynomial Regression in Mobile Devices," *IEEE Transactions on Consumer Electronics*, 61(1):63-71, February 2015.

[4] Bernd Bruegge and Allen H. Dutoit, *Object-Oriented Software Engineering using UML, Patterns and Java,"* Third Edition, Pearson Education, Inc., Prentice Hall, 2010.

[5] Han Dongmei, "Application Research of Rough-GA-BP Method in the Real Estate Early-Warning System," International Conference on Electrical and Control Engineering (ICECE), pp.483-485, 25-27 June 2010.

[6] Herbert A. Edelstein, *Introduction to Data Mining and Knowledge Discovery*, Third Edition, Two Crows Corporation, 1999.

[7] Usama M. Fayyad, Gregory Piatetsky-Shapiro, Padhraic Smyth, and Ramasamy Uthurusamy, *Advances in Knowledge Discovery and Data Mining*, MIT Press, 1996.

[8] Paolo Gastaldo, Alessio Leoncini, and Rodolfo Zunino, "Efficient Digital Implementation of Extreme Learning Machines for Classification," *IEEE Transactions on Circuits and Systems II: Express Briefs*, 59(8):496-500, Aug. 2012.

[9] Gavin Hackeling, *Mastering Machine Learning with Scikit-Learn*, First Edition, Packt Publishing, 2014.

[10] Peter Harrington, *Machine Learning in Action (First Edition)*, Manning Publications, 2012.

[11] Lilian Hobbs and Susan Hillson, *Oracle 8i Data Warehousing,* Digital Press, 2000.

[12] Markus Hofmann and Ralf Klinkenberg, *RapidMiner Data Mining Use Cases and Business Analytics Applications*, First Edition, CRC Press, 2014.

[13] Yongzhou Hou and Chengdong Yi, "Housing Market Dynamics in Urban China," International Conference on E -Business and E -Government (ICEE), pp. 1-4, 6-8 May 2011.

[14] Yating Hsu and David Lee, "Machine Learning for Implanted Malicious Code Detection with Incompletely Specified System Implementations," 2011 19th IEEE

International Conference on Network Protocols, Vancouver, BC, pp. 31-36, 2011.

[15] http://www.mris.com/ 2016.

[16] Claudia Imhoff, Nicholas Galemmo, and Jonathan G. Geiger, *Mastering Data Warehouse Design,* John Wiley & Sons, 2003.

[17] Nurlaila Ismail, Mohd H. Rahiman, Mohd N. Tiab, Nor Azah M. Ali, Mailina Jamil, and Saiful N. Tajuddin, "The Grading of Agarwood Oil Quality using k-Nearest Neighbor (k-NN)," 2013 IEEE Conference on Systems, Process & Control (ICSPC), pp. 1-5, 13-15 Dec. 2013.

[18] Baffoe-Bonnie John, "The Dynamic Impact of Macroeconomic Aggregates on Housing Prices and Stock of Houses: A National and Regional Analysis," *The Journal of Real Estate Finance and Economics*, 17(2):179-197, 1998.

[19] Yu-hong Kang and Li Xiang, "Research on Financial Distress Prediction of China Real Estate Public Companies Based on Z-Score Model," International Conference on Management Science and Engineering (ICMSE), pp. 1166-1173, 13-15 Sept. 2011.

[20] Ahmed Khalafallah, "Neural Network Based Model for Predicting Housing Market Performance," *Tsinghua Science and Technology*, 13(51):325-328, Oct. 2008.

[21] James. E. King, Samuel C. E. Jupe and Philip. C. Taylor, "Network State-Based Algorithm Selection for Power Flow Management Using Machine Learning," *IEEE Transactions on Power Systems*, 30(5):2657-2664, Sept. 2015.

[22] Willi Klosgen and Jan.M. Zytkow, *Handbook of Data Mining and Knowledge Discovery*, Third Edition*,* Oxford University Press, 2002.

[23] B. Lawrence, Kenneth Smith, T. Rosen, and George Fallis, "Recent Developments in Economic Models of Housing Markets," *Journal of Economic Literature,* 26(1):29-64, 1988.

[24] Kyong H. Lee and Naveen Verma, "A Low-Power Processor with Configurable Embedded Machine-Learning Accelerators for High-Order and Adaptive Analysis of Medical-Sensor Signals," *IEEE Journal of Solid-State Circuits*, 48(7):1625-1637, July 2013.

[25] Da-Ying Li, Wei Xu, Hong Zhao, and Rong-qiu Chen, "A SVR Based Forecasting Approach for Real Estate Price Prediction," *International Conference on Machine Learning and Cybernetics*, 2:970-974, 12-15 July 2009.

[26] Wanqing Li, Yong Zhao, Wenqing Meng, and Shipeng Xu, "Study on the Risk Prediction of Real Estate Investment Whole Process Based on Support Vector Machine," Knowledge Discovery and Data Mining, Second International Workshop, pp. 167-170, 23-25 Jan. 2009.

[27] Zhang Xiao Li, "Using Fuzzy Neural Network in Real Estate Prices Prediction," Control Conference, Chinese, pp. 399-402, June 31-July 26, 2007.

[28] Kang-yin Lu, Jin-xia Zhu, and Man-xue Chen, "An Empirical Study on the Relationship between Real Estate Price and Residents' Income Gap," International Conference on Management Science & Engineering (ICMSE), pp. 1860-1868, 17-19 Aug. 2014.

[29] Sumitra S. Nair, Robert M. French, Davy Laroche, and Elizabeth Thomas, "The Application of Machine Learning Algorithms to the Analysis of Electromyographic Patterns from Arthritic Patients," *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, 18(2):174-184, April 2010.

[30] Stephen Pudney and Limin Wang, "Housing Reform in Urban China: Efficiency Distribution and the Implications for Social Security," *Economics*, 62:141-159, 1995.

[31] Jonathan Schachter and Pierluigi Mancarella, "A Short-Term Load Forecasting Model for Demand Response Applications," 11th International Conference on European Energy Market (EEM), pp. 1-5, 28-30 May 2014.

[32] You Xue-Shi, Tian Jin-Xin, and Li Jian, "Based on Multiple Linear Regression Analysis of the Macroeconomic Factors Affecting Real Estate Prices," Information Systems for Crisis Response and Management (ISCRAM*)*, pp. 41-44, 25-27 Nov. 2011

[33] Ian Sommerville, *Software Engineering*, Ninth Edition, Addison-Wesley, 2001.

[34] Shao-rong Sun and Si-rong Ren, "System Dynamic Model of Housing Price Fluctuations Based on the Impact of Real Estate Tax Institution and Trading Behavior," Fifth International Conference on Business Intelligence and Financial Engineering (BIFE), pp. 130-134, 18-21 Aug. 2012.

[35] Akhil Wali, "*Clojure for Machine Learning*, First Edition*,* Packt Publishing, 2014.

[36] Shouyi Wang, Wanpracha Chaovalitwongse, and Robert Babuska, "Machine Learning Algorithms in Bipedal Robot Control," *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 42(5):728-743, Sept. 2012.

[37] Ting Wang, Yan-Qing Li, and Shu-Fei Zhao, "Application of SVM Based on Rough Set in Real Estate Prices Prediction," 4th International Conference on Wireless and Communications, Networking and Mobile Computing, pp. 1-4, 12-14 Oct. 2008.

[38] Ting Wang and Yanqing Li, "Application of SVM Based on Rough Set in Real Estate Investment Environment Comprehensive Evaluation," International Conference on Risk Management & Engineering Management, pp. 644-647, 4-6 Nov. 2008.

[39] Wuri Wedyawati and Meiliu Lu, "Mining Real Estate Listings using Oracle Data Warehousing and Predictive Regression," *Proceedings of the 2004 IEEE International Conference on Information Reuse and Integration,* pp.296-301, 8-10 Nov. 2004.

[40] Guo Wei, Cao Meiyan, and Zheng Jianfeng, "Study on Chinese Banks of Credit Risk Evaluation Models of Real-Estate Based on the BP-Neural Network Model," *2009 WRI World Congress on Computer Science and Information Engineering*, 2:288-292, March 31-April 2, 2009.

[41] William Wheaton and Gleb Nechayev, “The 1998-2005 Housing ‘Bubble’ and the Current ‘Correction’: What’s Different This Time?” *Journal of Real Estate Research*, 30(1):1-26, 2008.
[42] Ian H. Witten, Eibe Frank, and Mark A. Hall, *Data Mining,* Third Edition, Morgan Kaufmann, 2011.
[43] Chih-Hung Wu, Chi-Hua Li, I-Ching Fang, Chin-Chia Hsu, Wei-Ting Lin, Chia-Hsiang Wu, “Hybrid Genetic-Based Support Vector Regression with Feng Shui Theory for Appraising Real Estate Price,” First Asian Conference on Intelligent Information and Database Systems, pp. 295-300, 1-3 April 2009.
[44] Chih-Hung Wu, Wei-Han Su, and Ya-Wei Ho, “A Study on GPS GDOP Approximation Using Support-Vector Machines,” *IEEE Transactions on Instrumentation and Measurement*, 60(1):137-145, Jan. 2011.
[45] Ming Xue and Changjun Zhu, “A Study and Application on Machine Learning of Artificial Intellligence” International Joint Conference on Artificial Intelligence, pp. 272-274, 25-26 April 2009.
[46] Zhang Xuefang and Ji Wei, “Study on the Risk Assessment of Real Estate Project Based on BP Neural Network,” 2nd IEEE International Conference on Information and Financial Engineering (ICIFE), pp. 535-537, 17-19 Sept. 2010.
[47] Muneki Yasuda and Kazuyuki Tanaka, “Approximate Learning Algorithm in Boltzmann Machines,” *Neural Computation*, 21(11):3130-3178, Nov. 2009.
[48] Liang Yunbin, “Basic Concept of China’s Real Estate Forecasting and Early Warning System’s Economic Indicators,” *Beijing Real Estate*, 11:18-20, 1995.
[49] Byoung-Tak Zhang, “Cognitive Learning and the Multimodal Memory Game: Toward Human-Level Machine Learning,” IEEE International Joint Conference on Neural Networks, (IEEE World Congress on Computational Intelligence), pp. 3261-3267, 1-8 June 2008.
[50] Yong Zhang, Peng Li, Yingyezhe Jin, and Yoosuck Choe, “A Digital Liquid State Machine with Biologically Inspired Learning and Its Application to Speech Recognition,” *IEEE Transactions on Neural Networks and Learning Systems*, 26(11):2635-2649, Nov. 2015.
[51] Li Zheng and Yu Bing, “Impacts of Land Supply Factors on Housing Price: An Hedonic Price Model on Chengdu, China,” International Conference on Management Science and Engineering, pp. 2056-2060, 5-7 Oct. 2006.

Timothy Oladunni is a certified realtor in the state of Mayland, United States with more than ten years of experience. He is presently a doctoral student in the Department of Computer Science, Bowie State University, United States. He has an undergraduate background in Electrical Engineering from Yaba College of Technonology, Lagos, Nigeria and a master’s degree from the department of Computer Science, Bowie State University. Timothy’s research interest is in machine learning, artificial intelligence distributed systems and software engineering.

Sharad Sharma is an Associate Professor in the Department of Computer Science at the Bowie State University. He has received a Ph.D in Computer Engineering from Wayne State University, Detroit, MI in 2006, M.S. from University of Michigan, Ann Arbor, MI in 2003, and B. Arch from Birla Institute of Technolgy, Mesra, India in 1999. Dr. Sharma is the Director of the Virtual Reality Laboratory at the Bowie State University. His research focuses on using virtual reality and augmented reality as as tool for learning, training, and education.

Creating a Probabilistic Model for WordNet

Lubomir Stanchev [*]
California Polytechnic State University
San Luis Obispo, California, 93407, USA

Abstract

We present a probabilistic model for extracting and storing information from WordNet and the British National Corpus. We map the data into a directed probabilistic graph that can be used to compute the conditional probability between a pair of words from the English language. For example, the graph can be used to deduce that there is a 10% probability that someone who is interested in dogs is also interested in the word "canine". We propose three ways for computing this probability, where the best results are achieved when performing multiple random walks in the graph. Unlike existing approaches that only process the structured data in WordNet, we process all available information, including natural language descriptions. The available evidence is expressed as simple Horn clauses with probabilities. It is then aggregated using a Markov Logic Network model to create the probabilistic graph. We experimentally validate the quality of the data on five different benchmarks that contain collections of pairs of words and their semantic similarity as determined by humans. In the experimental section, we show that our random walk algorithm with logarithmic distance metric produces higher correlation with the results of the human judgment on three of the five benchmarks and better overall average correlation than the current state-of-the-art algorithms.

Key Words: Semantic similarity, probability-based semantic similarity and distances, Markov logic network for representing WordNet data, semantic similarity benchmarks for WordNet.

1 Introduction

Tens of scientists have spent decades to develop WordNet [22]. This word corpus contains very accurate information about 150,000 word forms from the English language and their senses. A *word form* is a word or a short phrase, such as "sports utility vehicle". Every word form can have multiple senses and every sense can be represented by multiple word forms. For example "a seat for one person" is the most popular sense of the word "chair".

The first problem that we will solve in this article is to show how to map the data from WordNet into a probabilistic graph. The graph will contain a node for each word form and each sense in WordNet. A directed edge between two nodes will be labeled with the probability that a user who is interested in the source node would also be interested in the destination node. For example, based on the definition of the first sense of the word "chair", we can create an edge between this sense and the word "seat". WordNet also contains information about the relationships between senses, where this information will also be used in creating the probabilistic graph.

The second problem that we will address in the article is how to measure the semantic similarity between two word forms in the graph. We will show two algorithms that consider disjoint paths between the two nodes. The first algorithm simply multiplies the weights of the edges along a path, while the second algorithm is more sophisticated and uses the Markov Logic Network (MLN) model [30]. The third algorithm is a Monte Carlo approximation algorithm that performs random walks in the graph.

The third and last problem that we will examine is how to experimentally validate the quality of the data in the graph and the quality of the semantic similarity algorithm using multiple benchmarks. As we will show in the next paragraph, the probabilistic graph has many applications. However, its usefulness is limited by the quality of the data in the graph. We will examine five benchmarks that have 201, 28, 65, 65, and 665 pairs of words, respectively. Each pair of words was given to multiple people and the average of the semantic similarity, as determined by their judgment, was recorded. We will compare the results of our three algorithms to that of 16 algorithms that form the current state-of-the-art in computing semantic similarity between words. Specifically, we will show that one of our algorithms produces higher correlation than the other 16 algorithms on three of the five benchmarks. Moreover, this algorithm has the highest average correlation over the five benchmarks.

The probabilistic graph has multiple applications. For example, in this article and in [43, 46], we focus on computing the degree of semantic similarity between a pair of word forms. In [45], we show how a probabilistic graph can be used to perform semantic search. This means that given a textual query, we can return documents that contain related words. For example, if we know that there is a 20% change that a

[*]Computer Science Department. Email: stanchev@gmail.com.

user who searches for cats will find documents that contain the word "pet" relevant, then we can return such documents as part of the query result. The documents that are returned are ranked based on the probability of being relevant to the input query, where the probabilistic graph can help us compute these probabilities. Lastly, [47] shows how a probabilistic graph can be used to perform semantic document clustering. For example, an online store can use the probabilistic graph to cluster the products that are offered in different categories. Two products should appear in the same category only when they have textual descriptions that contain words that are similar based on the semantic similarity distance that can be computed from the probabilistic graph.

Note that all the applications of the probabilistic graph rely on an efficient and precise algorithm for computing the conditional probability of a word form in the graph being relevant given that a different word form in the graph is relevant. The semantic similarity between two nodes can be computed as a function of the average of the probability of the first word form being relevant given that the second word form is relevant and the reverse. Since most of the relationships in the graph are between senses and not between word forms, we will present an algorithm that explores all the paths between two word form nodes in the graph in order to compute the conditional probability of the second word form being relevant given that the first word form is relevant.

Converting WordNet in a computer-friendly format is a daunting task because WordNet contains heterogeneous data. While there are a plethora of algorithms that process structured information [15, 37] and textual information [3, 16], experimental results have shown that processing both types of information yields the best results (e.g., [44]). The fact that processing natural language is intrinsically hard for computers makes the problem even harder. Although significant effort has been put in automated natural language processing (e.g., [9, 10, 25]), current approaches fall short of understanding the precise meaning of human text. In fact, the questions of whether computers will ever become as proficient as humans in understanding natural language text is an open problem. Lastly, note that the problem of computing the conditional probability of a word form in the graph being relevant given that a different word form in the graph is relevant is not trivial. As [46] shows, we can use the MLN model to compute the conditional probability along a single path. However, when there are multiple interweaving paths between the two nodes, exact computation of the conditional probability becomes computationally intractable.

To the best of our knowledge, our previous research in the area [46, 43, 45] is the only study on how structured and unstructured information from WordNet can be combined to capture the semantic relationship between word forms in a probabilistic model. Other approaches that extract information from WordNet (e.g., [18, 15, 37]) only consider the structured information in WordNet. However, we believe it is beneficial to capture all the information in WordNet, including the natural language text descriptions for the definition and example use of senses. Most existing research does not consider this information because natural language text is intrinsically hard to process.

The algorithm for creating the probabilistic graph first examines WordNet and creates a node for each word form and each sense. The label of a word form node is the word form and the label of a sense node is the definition of the sense. Next, we represent the relationship between nodes using logical formulas with weights. Following the MLN approach, the weight of a formula is equal to the natural logarithm of the odds of the formula being true. We slightly modify this expression to ensure that all the weights are positive. Our probability space consists of a random variable for each node in the graph and a single predicate called *rel* (stands for relevant). For example, we can model the relationship between the main sense of the word chair ("a seat for one person") and the first word in the definition of the sense using the Horn clause $rel(a\, seat\, for\, one\, person) \Rightarrow rel(seat)$. A weight will also be assigned to the formula and it will be based on how strongly we believe that someone who is interested in the sense will also be interested in the first non-noise word in its definition. After all the formulas are created, we draw edges between each pair of nodes that participate in a formula. We use the MLN model to aggregate the evidence about the conditional probability for each of these pairs. The resulting weight of an edge between two nodes is a normalized probability value that assures that the sum of the weights of all the edges that leave each node add up to one.

This article presents three algorithms for computing the conditional probability that a node is relevant given that a different node in the graph is relevant. Computing this probability using the MLN model without approximating the result is possible, but computationally intractable. Although [46] shows how to compute this probability along a single path, we are not aware of a practical algorithm that computes the probability when there are interweaving paths between the two nodes. In this article, we introduce a randomized Monte Carlo algorithm that performs multiple random walks, where the algorithm contains parameters for tuning the expected accuracy of the result.

In what follows, in Section 2 we cover related research. In Section 3, we present an overview of WordNet and our algorithm for creating the probabilistic graph. The main contributions of the article are in the next two sections. In Section 4, we present two existing algorithms and a novel Monte Carlo algorithm for computing the conditional probability between two nodes in the probabilistic graph. Section 5 shows previously unpublished experimental results that test the quality of the data in the probabilistic graph and the accuracy of the different algorithms for finding the conditional probability between two nodes on five different benchmarks. Lastly, Section 6 summarizes the article and suggests avenues for further research.

2 Related Research

First, note that this article builds on several previous papers by the same author. The paper [43] proposes how to build a similarity graph, where the weights of the edges in the graph correspond to the degree of directional semantic similarity between the nodes. However, the weight of the edges are not probabilities in the strict sense. The paper [46] extends this working by showing how to use the MLN model to convert the weights of the edges in the graph to strict probabilities. This article extends [46] by presenting more detailed introduction and related research sections. The algorithm for computing the probabilistic graph is slightly modified. However, the most important contribution is the new Monte Carlo algorithm for computing the conditional probability between two nodes in the probabilistic graph and the new experimental results that test the quality of the proposed model on five independent benchmarks and compare the results to 16 algorithms that form the current state-of-the-art in algorithms that compute semantic word similarity.

Existing research that applies Bayesian networks to represent knowledge deals with the uncertain or probabilistic information in the knowledgebase [26, 23]. Our approach slightly differs because we do not store the probability that a word form is relevant given that an adjacent word form in the graph is unrelated. We only store a single number along every edge (the conditional probability that the destination concept is relevant given that the source concept is relevant) and we do not store all the information that is needed to create the full joint distribution of the word forms. Our model is more compact and, as we will show in the experimental section, contains high quality data.

The idea of creating a graph that stores the degree of semantic similarity between word forms is not new. For example, Simone Ponzetto and Michael Strube show how to create a graph that only represents inheritance of words [15, 37]. Specifically, [28] proposed one of the first models that computes the information content by counting the number of occurrences of different words in the WordNet hierarchy. Alternatively, Glen Jeh and Jennifer Widom show how to approximate the similarity between words based on information about the structure of the graph in which they appear [13]. These papers, however, differ from our approach because we suggest representing available evidence from all type of sources, including natural language descriptions. Our approach is also different from the use of a semantic network [48] because the latter does not assign weights to the edges of the graph.

In this article, we show a method that uses the probabilistic graph to measure the semantic similarity between word forms. However, there are alternative methods to measure the semantic similarity between word forms. The most notable approach is the Google approach [6] in which the similarity between two word forms is measured as a function of the number of Google results that are returned by each word form individually and the two word forms combined. Note that there is a second relevant paper by Google research [20]. The paper explains how input text can be used to train a two-layer neural network. Once trained, the neural network can be used to predict what words will appear together in a text. This differs from our approach because we are interested in the semantic similarity between words, where similar words do not necessarily appear in the same sentence.

Other approaches that rely on data from the Internet include papers by Danushka Bollegala, Yutaka Matsuo, and Mitsuru Ishizuka [3] and by Swarnim Kulkami and Doina Caragea [16]. The first paper searches for lexicographical patterns between the words using a search engine. For example, in order to compute the similarity between the words "dog" and "cat", the system will search the Internet for the phrase "dog is a cat", among others. The second paper uses the Internet to create a *concept cloud* around each word and then computes the semantic distance between two words as a function of the distance between their concept clouds. For example, the word "feline" is part of the concept cloud for the word "cat". Although these approaches produce good measurement of semantic similarity, they have their limitations. First, they do not make use of structured information, such as the hyponym (i.e., is-a) relationship in WordNet. Second, they do not provide evidence about the strength of the relationship between the two word forms that are compared. In contrast, our approach can show the paths in the probabilistic graph between the two word forms, which serves as evidence that supports the similarity score.

Since the early 1990s, research on LSA (stands for *latent semantic analysis*) has been carried out [7]. The approach has the advantage of not relying on external information. Instead, it considers the closeness of word forms in text documents as proof of their semantic similarity. For example, LSA can be used to detect words that are synonyms [17]. This differs from our approach because we do not consider the closeness of the words in a document. For the most part, we process natural language text as a bag of terms, where the main exception is that we consider the order of the words in the definition of a WordNet sense when we create the logical formulas. The reason is that we assume that the first words in the definition of a sense are more important. The other difference is that our algorithm can extract overlapping terms from a text source. Although the LSA approach has its applications, we believe that using a high-quality word corpus, such as WordNet, is beneficial. Note as well that the LSA approach cannot be directly used to process structured knowledge.

Research from information retrieval is also relevant to creating and using the probabilistic graph. For example, if the word "ice" appears multiple times in the definition of one of the senses of the word "hockey", then this provides evidence about the relationship between the two words. Our approach uses a model that is similar to TF-IDF [14] (stands for term frequency – inverse document frequency) to compute the strength of the relationship. In the TF-IDF model, if the word "ice" appears two times in the definition of one of the senses of the word "hockey", then the term frequency can be computed as two. This number is multiplied by a number that is inversely proportional to how often the word "ice" appears in the definition of other senses. For example, if most senses contain the word "ice" as part of their definition, then the fact that one of the senses of the word "hockey" contains this word in its definition is inconsequential. Conversely, if the word "ice" appears only in the definition of a few senses, then the fact that the definition of one of the senses of the word "hockey" contains the word "ice" in its definition is statically meaningful.

Note that a plethora of research effort has recently focused on using a description language, such as the *ontology web language* (OWL) [51], to describe resources. A semantic query language, such as SPARQL [39] (a recursive acronym that stands for SPARQL Protocol and RDF Query Language), can be used to search for relevant items. This research differs from our approach to semantic search in [45] because it does not provide ranking of the query result. At the same time, a SPARQL query returns exactly the resources that fulfill the query description. Alternatively, [45] returns resources that are related to the input query in ranked order. There is no need to describe the resources using a mathematical language, there is no need to phrase the query using a mathematical language, and the system is much more scalable (OWL knowledgebases are usually applied only to a limited knowledge domain because query answering over them is intrinsically computationally expensive.) Lastly, there are papers that consider a hybrid approach for information retrieval using both an ontology and keyword matching. For example, [32] examines how queries can be expanded based on the information from an OWL knowledgebase. Alternatively, [49] proposes a ranking function that depends on the length of the logical derivation of the result, where the assumption is that shorter derivations will produce more relevant documents. Unfortunately, these approaches are only useful in the presence of an ontology and research on automatic annotation of resources with OWL descriptions is still in its early stages of development.

There has also been research in the area of combining a subset of OWL called RDF [27] (stands for Resource Description Framework) with information retrieval approaches, such as BM25F [31] (a version of the TF-IDF approach). For example, [2] shows how to use natural language to query RDF stores. Note that this is a keywords-matching search approach and it does not take into account that the same query can be phrased differently using different words and terms. There have also been several papers that explore how to rank the result of queries over RDF data. For example, [5] uses the TF-IDF algorithm to rank the result of an RDF query.

Lastly, note that the probabilistic graph can be applied to the problem of query expansion in natural language search systems [38]. For example, a user may search for "Mediterranean Restaurants". A smart search engine needs to expand the search query and also search for Egyptian, Moroccan, Syrian, and Turkish restaurants, among others. This expansion is based on the knowledge in the probabilistic graph.

3 Building the Probabilistic Graph

3.1 About WordNet

WordNet [22] gives us information about the words in the English language. In our study, we use WordNet 3.0, which contains approximately 150,000 different words. WordNet also contains phrases, such as "sports utility vehicle". WordNet uses the term *word form* to refer to both the words and the phrases in the corpus. Note that the meaning of a word form is not precise. For example, the word "spring" can mean the season after winter, a metal elastic device, or the natural flow of ground water, among others. This is the reason why WordNet uses the concept of a *sense*. For example, earlier in this paragraph we cited three different senses of the word "spring". Every word form has one or more senses and every sense is represented by one or more word forms. A human can usually determine which of the many senses a word form represents by the context in which the word form is used.

WordNet contains a plethora of information about word forms and senses. For example, it contains the definition and example use of each sense. Consider the word "chair". One of its senses has the definition: "a seat for one person, with a support for the back" and the example use: "he put his coat over the back of the chair and sat down". Two other senses of the word have the definitions: "the position of a professor" and "the officer who presides at the meetings of an organization". We will process these textual descriptions to extract evidence about the strength of the relationship between the initial word forms and the word forms that appear in the definition and example use of their senses. Note that WordNet also provides information about the frequency of use of each sense. This represents the popularity of the sense in the English language relative to the popularity of the other senses of the word form. For example, the first sense of the word "chair" (a seat for one person, with a support for the back) is given a frequency of 35, the second sense (the position of a professor) is given frequency of just two, while the third sense (the officer who presides at the meetings of an organization) is given a frequency of one.

WordNet also contains information about the relationship between senses. The senses in WordNet are divided into four categories: nouns, verbs, adjectives, and adverbs. For example, WordNet stores information about the hypernym and hyponym relationships between nouns. The *hypernym* relationship corresponds to the "kind-of" relationship. For example, "canine" in a hypernym of "dog". The *hyponym* relationship is the reverse. For example, "dog" is a hyponym of "canine". WordNet also provides information about the meronym and holonym relationships between noun senses. The *meronym* relationship corresponds to the "part-of" relationship. Note that WordNet provides three types of meronyms: *part*, *member*, and *substance*. The three types of meronyms can be explained with the following examples: a "tire" is part of a "car", a "car" is a member of "traffic jam", and a "wheel" is made from "rubber", respectively. The *holonym* relationship is the reverse of the meronym relationship. For example, "building" is a holonym of "window". For verbs, WordNet defines the *hypernym* and *troponym* relationships. X is a hypernym of Y if performing X is one way of performing Y. For example, "to perceive" is a hypernym of "to listen". The verb Y is a troponym of the verb X if the activity Y is doing X in some manner. For example, "to lisp" is a troponym of "to talk". Lastly, WordNet defines the *related to* and *similar to* relationships between adjective senses, which are self explanatory. We will use all this structured information from WordNet as evidence about the degree of conditional probability between senses.

3.2 The Probabilistic Model

We create a random variable for each sense and each word form in WordNet. We will refer to a random variable by its label, where the label of a word form variable is the word form and the label of a sense variable is the definition of the sense. In order to avoid ambiguity, we convert all labels to lower case. In this model, each random variable will have a string label and no two random variables will have the same label.

We add a single predicate to the model. The name of the predicate is *rel* and it tells us if a word form or sense is relevant in the current world. Our model contains only logical formulas that are Horn clauses of the form: $rel(X) \Rightarrow rel(Y)$. We will add a weight to each logical formula, where the weight will be computed using the following expression.

$$w(rel(X) \Rightarrow rel(Y)) = ln(\frac{P^{+}(Y|X)}{1-P^{+}(Y|X)}) \tag{1}$$

Following the MLN model [30], the weight of a logical formula is equal to the natural logarithm of the odds of the formula being true, that is $ln(\frac{p}{1-p})$. However, this will allow formulas with negative weights, which is undesirable. When aggregating evidence, a MLN works by interpreting formulas with positive weights as positive reinforcement and formulas with negative weights as evidence why the formula does not hold. By making all weights positive, we ensure that all the formulas will have a positive contribution to the aggregated conditional probability between two concepts. Note that when we say that there is a 10% probability that the word "table" is relevant given that the word "chair" is relevant, we want this evidence to increase the conditional probability of the word "table" being relevant given the word "chair" is relevant. We make the weights positive by performing a linear transformation of the probability to the range $[0.5, 1]$. Specifically, we define P^{+} as follows.

$$P^{+}(Y|X) = 0.5 + \frac{P^{e}(Y|X)}{2} \tag{2}$$

We use $P^{e}(Y|X)$ to denote our confidence of the formula being true and refer to this value as the *evidence probability*. For example, if we know that the evidence probability is 0.10 (i.e., we are 0.10 confident that someone who is interested in the word "chair" will also be interested in the word "table"), then $P^{+}(table|chair) = 0.55$ and the weight of the formula will be calculated as $ln(0.55/0.45) = 0.2$.

Note that the same formula can appear multiple times in our knowledgebase, but possibly with different weights. At the end of this section, we will show how we can apply the MLN model to aggregate multiple evidence about the conditional probability between two concepts. Before that, we describe an algorithm that models WordNet as a set of Horn clauses with weights. Note that, for the most part, we will only describe how to compute the evidence probability, where the weight of each formula can be computed using Equations 1 and 2.

3.3 Processing the Senses

We first show how to create logical formulas that show the relationship between a word form and all its senses. Consider the word chair and its three meanings: "a seat for one person", "the position of a professor" and "the officer who presides at meetings". Suppose that WordNet gives a frequency of 35, 2, and 1, respectively, for the three senses. We will then crate the following formulas and probabilities.

$$rel(chair) \Rightarrow rel(a\,seat\,for\,one\,person), (35/38)$$
$$rel(chair) \Rightarrow rel(the\,position\,of\,a\,professor), (2/38)$$
$$rel(chair) \Rightarrow rel(the\,officer\,who\,presides\,at\,meetings), (1/38)$$

Note that the word "chair" has three meanings. Based on the frequencies that we are given, the evidence probabilities for the three relationships are 35/38, 2/38, and 1/38, respectively. Note that for each formula, we put the evidence probability in parentheses. We can then compute the weight of the formula using Equations 1 and 2. When we assign an actual weight to a formula, we omit the parenthesis around the number. In general, we will compute the evidence probability as the frequency of the sense divided by the sum of the frequencies of all the senses for the word form. Here is the general formula, where $\{sense_i\}_{i=1}^{n}$ are all the senses of the word form.

$$rel(word\,form) \Rightarrow rel(sense\,of\,the\,word\,form), \\ (\frac{frequency(sense)}{\sum_{i}^{n} frequency(sense_i)}) \tag{3}$$

In our example, we will also add the following formulas and weights. Since there are no parentheses, the expressions show weights and not evidence probabilities.

$$rel(a\,seat\,for\,one\,person) \Rightarrow rel(chair), 10$$
$$rel(the\,position\,of\,a\,professor) \Rightarrow rel(chair), 10$$
$$rel(the\,officer\,who\,presides\,at\,meetings) \Rightarrow rel(chair), 10$$

In general, we always add a formula with weight 10 between a sense and all the word forms that it represents. The general formula is shown next.

$$rel(sense\,of\,a\,word\,form) \Rightarrow rel(word\,form), 10 \tag{4}$$

The reason for this formula is that we have a very high degree of confidence that if a sense is relevant, then so are all the word forms that represent the sense. A weight of 10 corresponds to evidence probability of above 99.99%. Note that in a MLN we cannot assign an evidence probability of one to a formula because this translates to a weight that is equal to infinity.

3.4 Processing the Definitions of the Senses

We next show how to model the relationship between a sense and the non-noise word forms in its definition. Note that our algorithm uses a list of about one hundred noise words, such as "who", "where", "at", "about" and so on. Consider the second sense of the word "chair": "the position of a professor". The noise words: "the", "of", and "a" will be ignored. We will therefore be left with two words: "position" and "professor". As a result, we will create the following formulas.

$$rel(the\,position\,of\,a\,profeesor) \Rightarrow rel(position), (0.6)$$
$$rel(the\,position\,of\,a\,profeesor) \Rightarrow rel(professor), (0.48)$$

The formulas represent the connection between a sense and the non-noise words in its definition. We assume that the first words in the definition of a sense are far more important than the later words. We will therefore multiply the probability by $coef = 1.0$ for the first non-noise word form and keep decreasing this coefficient by 0.2 for each sequential word form until the value of the coefficient reaches 0.2. We compute the evidence probability of each formula using the equation $coef * minMax(0, 0.6, ratio)$, where the variable $ratio$ is calculated as the number of times the word form appears in the definition of the sense divided by the total number of non-noise words in the sense.

$$rel(sense) \Rightarrow rel(word\,form\,in\,the\,sense\,definition), (coef * minMax(0, 0.6, \frac{frequency\,of\,word\,form}{sum\,of\,frequencies})) \quad (5)$$

The third parameter of the *minMax* function expresses the importance of the word form in the definition of the sense. For example, if there are only two word forms in the definition of the sense, then they are both very important. However, if there are 20 word forms in the definition of the sense, then each individual word form is less important. The *minMax* function makes the difference between the two cases less extreme. Using this function, the evidence probability of the formula in the second case will be only roughly four times smaller than the evidence probability of the formula in the first case. This is a common approach when processing text. The importance of a word in a text decreases as the size of the text increases, but the importance of the word decreases at a slower rate than the rate of the growth of the text. We use the *minMax* function every time we compare the number of occurrences of a word form in a document compared to the total number of words in the document.

The *minMax* function returns a number that is in most cases between the first two arguments, where the magnitude of the number is determined by the third argument. Since the appearance of a word form in the definition of a sense is not a reliable source of evidence about the relationship between the word form and the sense, the value of the second argument is set to 0.6. The constant 0.6 is related to the probability that someone who is interested in a sense will also be interested in one of the word forms in the definition of the sense. Note that throughout this paper we introduce multiple constants. In [44], we give experimental evidence why these constants are meaningful and produce good results.

Formally, the *minMax* function is defined as follows.

$$minMax(minValue, maxValue, ratio) = minValue + (maxValue - minValue) * \frac{-1}{log_2(ratio)}$$

Note that when $ratio = 0.5$, the function returns $maxValue$. An unusual case is when the value of the variable $ratio$ is bigger than 0.5. For example, if $ratio = 1$, then we have division by zero and the value for the function is undefined. We handle this case separately and assign value to the function equal to $1.2 * maxValue$. This is an extraordinary case when there is a single non-noise word in the text description and we need to assign higher evidence probability to the formula.

In our example, $ratio = \frac{1}{2}$ and therefore $minMax(0, 0.6, ratio) = 0.6$. Therefore, the evidence probability of the first formula is $coef * 0.6 = 1 * 0.6 = 0.6$ and for the second formula: $coef * 0.6 = 0.8 * 0.6 = 0.48$. To summarize, we assume that the probability that a user is interested in a word form will be higher if : (1) the word form appears multiple times in the definition of the sense, (2) the word form is one of only few words in the definition of the sense, and (3) the word form is one of the first word forms of the definition of the sense.

3.5 Processing the Example Uses of a Senses

WordNet also includes example uses for each sense. In this subsection, we show how to represent this information as formulas with weights. For example, in WordNet the sentence "he put his coat over the back of the chair and sat down" is shown as an example use of the first sense of word "chair". Since the example use represents evidence that is weaker than the evidence from the definition of a sense, we will calculate the evidence probability as $minMax(0, 0.2, ratio)$. Here, the variable $ratio$ is the number of times the word form appears in the example use divided by the total number of non-noise words in the example use. The constant 0.2 is related to the probability that someone who is interested in a sense will be also interested in one of the word forms in the example use of the sense. The following formulas are created from the first sense of the word "chair" and its example use. Note that the noise words have been omitted.

$$rel(a\,seat\,for\,one\,person) \Rightarrow rel(put),\ (0.09)$$
$$rel(a\,seat\,for\,one\,person) \Rightarrow rel(coat),\ (0.09)$$
$$rel(a\,seat\,for\,one\,person) \Rightarrow rel(back),\ (0.09)$$
$$rel(a\,seat\,for\,one\,person) \Rightarrow rel(sat),\ (0.09)$$
$$rel(a\,seat\,for\,one\,person) \Rightarrow rel(down),\ (0.09)$$

The evidence probability is the same for all edges because all words appear once in the example use. For all words, the value of *ratio* is equal to $\frac{1}{5}$. Unlike the case with the definition of a sense, the first words in the example use are not considered to be more important. Therefore, we ignore the order of the words in the example use of a sense. The precise calculation for the evidence probability is $0.2 * (\frac{-1}{log_2(0.2)}) = 0.09$. The general formula is shown next.

$$rel(sense) \Rightarrow rel(word\,form\,in\,the\,example\,use\,of\,the\,sense),$$
$$(coef * minMax(0, 0.2, \frac{frequency\,of\,word\,form\,in\,example\,use}{sum\,of\,frequencies})) \tag{6}$$

3.6 Processing the Backward Relationships

We also create formulas for the probability that a sense is relevant given that a word form that appears in its definition is relevant. The evidence probability of the formula is computed as $minMax(0, 0.3, ratio)$, where the variable *ratio* is the number of times the word form appears in the definition of the sense divided by the total number of occurrences of the word form in the definition of all senses. The constant 0.3 relates to the probability that someone who is interested in a word form will also be interested in one of the senses that have the word form in their definition. Here, we assume that the backward relationship is not as strong as the forward relationship. As an example, if the word "position" occurs as part of the definition of only three senses and exactly once in each definition, then we will add the following formula for the second sense of the word "chair". The evidence probability is computed as $minMax(0, 0.3, ratio) = 0.3 * \frac{-1}{log_2(\frac{1}{3})} = 0.19$ and the formula is as follows.

$$rel(position) \Rightarrow rel(the\,position\,of\,a\,professor),\ (0.19)$$

The general formula is shown next.

$$rel(a\,word\,form\,in\,a\,sense) \Rightarrow rel(sense), (minMax(0, 0.3,$$
$$\frac{frequency\,of\,word\,form\,in\,sense\,definition}{sum\,of\,frequencies}) \tag{7}$$

Similarly, we will create a formula that shows the conditional probability between a word form and a sense that contains the word form in its example use. The weight of an edge in this case will be computed as $minMax(0, 0.1, ratio)$. Here, the *ratio* parameter is the number of times the word form appears in the example use of the sense divided by the total number of occurrences in the example uses of all senses. The constant 0.1 relates to the probability that someone who is interested in a word form will also be interested in one of the senses that have the word form in their example use. This value is smaller than the value for the definition of a sense because the words in the definition of a sense are more closely related to the meaning of the sense. As an example, if the word "coat" occurs as part of the example use of only three senses and exactly once in each sense, then we will add the following formula for the first sense of the word "chair". The evidence probability is computed as $minMax(0, 0.1, \frac{1}{3}) = 0.1 * \frac{-1}{log_2(\frac{1}{3})} = 0.06$. Recall that the example use of this sense is: "he put his coat over the back of the chair and sat down".

$$rel(coat) \Rightarrow rel(a\,seat\,for\,one\,person),\ (0.06)$$

The general formula is shown next.

$$rel(a\,word\,form\,in\,the\,example\,use\,of\,a\,sense) \Rightarrow rel(sense),$$
$$minMax(0, 0.1, \frac{frequency\,of\,word\,form\,in\,example\,use}{sum\,of\,frequencies}) \tag{8}$$

3.7 Populating the Frequencies of the Senses

So far, we have shown how to extract information from textual sources, such as the text for the definition and example use of a sense. We will next show how structured knowledge, such as the hyponym (a.k.a. kind-of) relationship between senses, can be represented as logical formulas. Most existing approaches [28] explore these relationships by evaluating the *information content* of different word forms. Here, we adjust this approach and focus on the frequency of use of each word in the English language as described in the University of Oxford's British National Corpus. The description of this corpus, as presented in [4], is: "The British National Corpus is a 100 million word collection of samples of written and spoken language from a wide range of sources, designed to represent a wide cross-section of British English, both spoken and written, from the late twentieth century."

Definition 1. *Let s be a sense. Let $\{wf_i\}_{i=1}^n$ be the word forms for that sense. We will use $BNC(wf)$ to denote the frequency of the word form in the British National Corpus. Let $p_s(wf)$ be the frequency of use of the sense s of the word form wf, as specified in WordNet, divided by the sum of the frequencies of use of all senses of wf (also as defined in WordNet). Then we define the* size *of s to be equal to $|s| = \sum_{i=1}^{n}(BNC(wf_i) * p_s(wf_i))$.*

The above formula approximates the size of a sense by looking at all the word forms that represent the sense and figuring out how much each word form contributes to the sense. The size of a sense approximates its popularity. For example, according to WordNet the word "president" has six different senses with frequencies: 14, 5, 5, 3, 3, and 1. Let us refer to the fourth sense: "The officer who presides at the meetings ..." as s. According to Definition 1, $p_s(president) = 3/31 =$

0.096 because the frequency of s is 3 and the sum of all the frequencies is 31. Since the British National Corpus shows the frequency of the word "president" as 9781, the contribution of the word "president" to the size of the sense s is equal to $|s| = BNC(president) * p_s(president) = 9781 * 0.096 = 938.98$. Other word forms that represent the sense s, such as "chairman", will also contribute to the size of the sense.

3.8 Processing Structured Knowledge About Nouns

WordNet defines the *hyponym* (a.k.a. kind-of) relationship between senses that represent nouns. For example, the most popular sense of the word "dog" is a hyponym of the most popular sense of the word "canine". Consider the first sense of the word "chair": "a seat for one person". WordNet defines 15 hyponyms for this sense, including senses for the words "armchair" and "wheelchair". We will add formulas that show the conditional probability between this first sense of the word "chair" and each of the hyponyms. Let the probability that someone who is interested in a sense is also interested in one of the sub-senses be equal to 0.9. This probability is high because, for example, someone who is interested in the first sense of the word "chair" is probably also interested in one of the chair types. In order to determine the evidence probability of each formula, we need to compute the size of each sense. In the British National Corpus, the frequency of "armchair" is 657 and the frequency of "wheelchair" is 551. Since both senses are associated with a single word form, we do not need to consider the frequency of use of each sense. If "armchair" and "wheelchair" were the only hyponyms of the sense "a seat for one person", then we need to add the following formulas.

$$\begin{aligned}&rel(a\,seat\,for\,one\,person) \Rightarrow\\&\quad rel(chair\,with\,support\,on\,each\,side\,for\,arms), (0.49)\\&rel(a\,seat\,for\,one\,person) \Rightarrow\\&\quad rel(a\,moveable\,chair\,mounted\,on\,large\,wheels), (0.41)\end{aligned}$$

The evidence probabilities were computed as $0.9 * 657/1208 = 0.49$ and $0.9 * 551/1208 = 0.41$. In general, the evidence probability is computed as 0.9 multiplied by the size of the sense and divided by the sum of the sizes of all the hyponym senses of the initial sense.

$$rel(sense) \Rightarrow rel(hyponym\,of\,the\,sense),\quad (0.9 * \frac{|hyponym\,of\,the\,sense|}{\sum_{s\,is\,a\,hyponym\,of\,the\,sense} |s|}) \tag{9}$$

The idea is that the conditional probability for "bigger" senses will be bigger because it is more likely that a bigger sense is relevant. Note that here we do not apply the *minMax* function. The reason is that the function is only relevant when computing the ratio of the number of occurrences of a word form in text relative to the size of the text.

We will also create formulas for the *hypernym* relationship (the inverse of the hyponym relationship). For example, the first sense of the word "canine" is a hypernym of the first sense of the word "dog". The evidence probability for each formula will be the same and equal to the constant 0.3. This represents the probability that someone who is interested in a sense will be also interested in the hypernym of the sense. For example, if a user is interested in the sense "wheelchair", then they may be also interested in the first sense of the word chair. However, this probability is not a function of the different hypernyms of the sense. Next, we show the formula from our example.

$$\begin{aligned}&rel(chair\,with\,support\,on\,each\,side\,for\,arms) \Rightarrow\\&\quad rel(a\,seat\,for\,one\,person), (0.3)\end{aligned}$$

The general formula is shown below.

$$rel(sense) \Rightarrow rel(hypernym\,of\,the\,sense), (0.3) \tag{10}$$

We next consider the *meronym* (a.k.a. part-of) relationship between nouns. Note that we do not make a distinction between the three types of meronyms (part, member, and substance) and process them identically. For example, WordNet contains information that the sense of the word "back": "a support that you can lean against ..." and the sense of the word "leg": "one of the supports for a piece of furniture" are both meronyms of the first sense of the word "chair". In other words, back and 'legs are building parts of a chair. Part of this information can be represented using the following equations.

$$\begin{aligned}&rel(a\,seat\,for\,one\,person) \Rightarrow\\&\quad rel(a\,support\,that\,you\,can\,lean\,against), (0.3)\\&rel(a\,seat\,for\,one\,person) \Rightarrow\\&\quad rel(one\,of\,the\,supports\,for\,a\,piece\,of\,furniture), (0.3)\end{aligned}$$

In general, we compute the evidence probability as $0.6/n$, where n is the number of meronyms of the sense. Here is the general formula.

$$rel(sense) \Rightarrow rel(meronym\,of\,the\,sense),\quad (\frac{0.6}{number\,of\,meronyms\,of\,the\,sense}) \tag{11}$$

The constant 0.6 represents the probability that a user who is interested in a sense of a word form is also interested in one of its meronyms. In our system, this coefficient is set to 0.6 because the meronym relationship provides weaker evidence than the hyponym relationship. The reasoning behind the formula is that the more meronyms a sense has, the less likely it is that we are interested in a specific meronym.

We also represent the *holonym* (a.k.a. contains) relationship. For example, the main sense of the word "building" is a holonym of the main sense of the word "window". Similar to hypernyms, we set the evidence probabilities for the holonym relationship to a constant. The constant is 0.15 because the holonym relationship is not as strong as the hypernym relation. For example, the fact that someone is interested in the first sense of the word "window" does not translate in strong confidence that they are also interested in the whole building. For our running example, we create the following formulas.

$$\begin{aligned}&rel(a\,support\,that\,you\,can\,lean\,against) \Rightarrow\\&\quad rel(a\,seat\,for\,one\,person),\,(0.15)\\&rel(one\,of\,the\,supports\,for\,a\,piece\,of\,furniture) \Rightarrow\\&\quad rel(a\,seat\,for\,one\,person),\,(0.15)\end{aligned} \tag{12}$$

The general formula is shown next.

$$rel(sense) \Rightarrow rel(holonym\,of\,the\,sense),\,(0.15)$$

3.9 Processing Structured Knowledge About Verbs

We will first represent the *troponym* (a.k.a. doing in some manner) relationship for verbs. For example, to lisp is a troponym of to talk. Suppose that the verb "talk" has only three troponyms: "lisp", "orate", and "converse". If the sizes of the main senses of the three verbs are 18, 1, and 95 (as determined by the formula in Definition 1), respectively, then we will create the following equations.

$$\begin{aligned}&rel(an\,exchange\,of\,ideas\,via\,conversation) \Rightarrow\\&\quad rel(talk\,with\,a\,lisp),\,(0.14)\\&rel(an\,exchange\,of\,ideas\,via\,conversation) \Rightarrow\\&\quad rel(talk\,pompously),\,(0.01)\\&rel(an\,exchange\,of\,ideas\,via\,conversation) \Rightarrow\\&\quad rel(carry\,on\,a\,conversation)\,(0.75)\end{aligned}$$

The left side of the formulas contains the first sense of the word "talk": "an exchange of ideas via conversation", while the right side of the formulas contains the senses for "lisp", "orate" and "converse". The first formula expresses the conditional probability between the senses for "talk" (an exchange of ideas via conversation) and "lisp". The evidence probability for the formula is equal to $0.9 * \frac{18}{114} = 0.14$. The constant 0.9 represents that there is a 90% chance that if someone is interested in a verb, then they are also interested in one of its troponyms. We arrive at the expression $18/114$ by dividing the size of the sense by the sum of the sizes of all the troponym senses. The general formula is shown next.

$$\begin{aligned}&rel(sense) \Rightarrow rel(troponym\,of\,the\,sense),\\&\quad(0.9 * \frac{|sense|}{\sum\limits_{s\,is\,a\,troponym\,of\,the\,sense} |s||})\end{aligned} \tag{13}$$

We will also add formulas for the reverse relationship with evidence probability of 0.3. For example, we will add the following formula.

$$rel(talk\,with\,a\,lisp) \Rightarrow rel(an\,exchange\,of\,ideas\,via\,\ldots),\,(0.3)$$

This means that if someone is interested in one of the troponyms, then there is a 30% chance that they are also interested in the original verb. The general formula is shown next.

$$rel(troponym\,of\,the\,sense) \Rightarrow rel(sense),\,(0.3) \tag{14}$$

The hyponym and hypernym relationships are defined not only for nouns, but also for verbs. The two relationships are the reverse of each other. In other words, if X is a hyponym of Y, then Y is a hypernym of X. The hypernym relationship for verbs corresponds to the "one way to" relationship. For example, the verb "perceive" is the hypernym of the verb "listen" because one way of perceiving something is by listening. As expected, the verb "listen" is a hyponym of the verb "perceive". The first sense of the word "perceive" is "to become aware of through the senses". Suppose that the first senses of the verbs "listen" and "see" are the only hypernyms of the verb "perceive".

We will assume that the probability that someone who is interested in a verb sense is also interested in one of the hyponym senses is equal to 0.9. This probability is high because, for example, someone who is interested in perceiving is probably also interested in one of the ways to perceive. In order to determine the evidence probabilities of the formulas, we need to compute the size of each sense. In the British National Corpus, the frequency of "listen" is 1241 and the frequency of "see" is 3624. Since both senses are associated with a single word form, we do not need to consider the frequency of use of each sense. If "perceive" and "see" were the only hyponyms of the sense "to become aware of thought and senses", then we will create the following formulas.

$$\begin{aligned}&rel(to\,become\,aware\,of\,thought\,and\,senses) \Rightarrow\\&\quad rel(pay\,attention\,to\,sound),\,(0.23)\\&rel(to\,become\,aware\,of\,thought\,and\,senses) \Rightarrow\\&\quad rel(percieve\,by\,sight),\,(0.67)\end{aligned}$$

The evidence probability for each formula is equal to 0.9 multiplied by the size of the sense and divided by the sum of the sizes of all the hyponym senses of the initial sense. For example, the evidence probability of the first formula is $0.9 * 1241/4865 = 0.23$ and the evidence probability of the second formula is $0.9 * 3624/4865 = 0.67$. The idea behind the formula is that the conditional probabilities to "bigger" senses will be bigger because it is more likely that they are relevant. The general formula is shown next.

$$\begin{aligned}&rel(sense) \Rightarrow rel(hyponym\,of\,the\,sense),\\&\quad(0.9 * \frac{|sense|}{\sum\limits_{s\,is\,a\,hyponym\,of\,the\,sense} |s|})\end{aligned} \tag{15}$$

We will use an evidence probability of 0.3 for the hypernym (the reverse of the hyponym) relationship. For example, the main sense of the verb "perceive" is a hypernym of the main senses of the verbs "listen" and "see". This information can be expressed using the following formulas.

$$\begin{aligned}&rel(pay\,attention\,to\,sound) \Rightarrow\\ &\quad rel(to\,become\,aware\,of\,thought\,and\,senses),\,(0.3)\\ &rel(percieve\,by\,sight) \Rightarrow\\ &\quad rel(to\,become\,aware\,of\,thought\,and\,senses),\,(0.3)\end{aligned}$$

The number 0.3 represents the probability that someone who is interested in a sense will also be interested in the hypernym of the sense. For example, if a user is interested in the sense "see", then they may be also interested in the first sense of the word perceive. However, this probability is not a function of the different hypernyms of the sense. The general formula is shown next.

$$rel(sense) \Rightarrow rel(hypernym\,of\,the\,sense),\,(0.3) \tag{16}$$

3.10 Processing Structured Knowledge About Adjectives

WordNet defines two relationships for adjectives: *related to* and *similar to*. For example, the first sense of the adjective "slow" has definition: "not moving quickly", while the first sense of the adjective "fast" has the definition: "acting or moving or capable of acting or moving quickly". WordNet specifies that the two senses are *related to* each other. We will represent this relationship using the following formulas.

$$\begin{aligned}&rel(not\,moving\,quickly) \Rightarrow rel(acting\,or\,moving\,quickly),\,(0.6)\\ &rel(acting\,or\,moving\,quickly) \Rightarrow rel(not\,moving\,quickly),\,(0.6)\end{aligned}$$

This represents that there is a 60% probability that someone who is interested in an adjective is also interested in a "related to" adjective. This probability is high because the "related to" relationship represents relatively strong semantic similarity. The general formula is shown below.

$$rel(sense) \Rightarrow rel(related\,to\,sense),\,(0.6) \tag{17}$$

WordNet also defines the *similar to* relationship between adjectives. We create formulas with evidence probability of 0.8 for this relationship because the "similar to" relationship is stronger than the "related to" relationship. In other words, we believe that there is an 80% probability that someone who is interested in an adjective is also interested in a "similar to" adjective. For example, WordNet contains the information that the sense for the word "frequent": "coming at short intervals" and the sense for the word "prevailing": "most frequent or common" are similar to each other. We will therefore create the following formulas.

$$\begin{aligned}&rel(coming\,at\,short\,intervals) \Rightarrow rel(most frequent\,\ldots),\,(0.8)\\ &rel(most frequent\,or\,common) \Rightarrow rel(coming\,at\ldots),\,(0.8)\end{aligned} \tag{18}$$

Note that both the "similar to" and "related to" relationships are symmetric and therefore the evidence probability for each formula and its reverse is the same. The general formula is shown next.

$$rel(sense) \Rightarrow rel(similar\,to\,sense),\,(0.6) \tag{19}$$

3.11 Building the Probabilistic Graph

Equations 3-19 from the previous subsections show how to create Horn clauses from WordNet. Once the formulas are extracted, they are converted into a probabilistic graph. In order to do so, first, we create a node for each random variable, that is, for each word form and each sense. Next, we convert the evidence probabilities of the formulas to weights using Equation 1 and 2. Note that there can be several identical formulas with possibly different weights that are generated. When this is the case, we will merge all such formulas into a single formula. The weight of the new formula is equal to the sum of the weights of the old formulas. For example, consider the following two formulas.

$$\begin{aligned}&rel(X) \Rightarrow rel(Y),\,2.3\\ &rel(X) \Rightarrow rel(Y),\,1.1\end{aligned} \tag{20}$$

The old formulas will be removed and the following new formula will be created.

$$rel(X) \Rightarrow rel(Y),\,3.4 \tag{21}$$

First, note that we are adding the weights of the formulas and not the probabilities and therefore the evidence probability of the formula will always stay below 1.0. Second, note that since the evidence probabilities are always above 0.5, our model is monotonic (i.e, adding a new formula will always increase the evidence probability of the final formula). Lastly, note that adding the weights is consistent with the MLN model. Specifically, the probability of a world X is computed using the following formula.

$$P(X) = \frac{1}{total} e^{(\sum_F w(F)*|F(X)|)} \tag{22}$$

In the formula, *total* is a normalizing constant that is used to make sure that all the probabilities over all worlds add up to one. The sum is over all formulas F in our knowledgebase. The expression $w(F)$ is used to denote the weight of the formula F and $|F(X)|$ is equal to one when the formula F is true in the world X and is equal to 0 otherwise. Obviously, merging identical formulas by adding up their weights follows the above formula.

Next, we add an edge between X and Y in the graph for each logical formula of the following type.

$$rel(X) \Rightarrow rel(Y),\,w$$

The weight of the edge will be converted to a probability and will be computed using the following formulas.

$$p = \frac{1}{1+e^{-w}}$$
$$edgeweight = \frac{2*p-1}{total}$$

The first formula converts the weight to a probability. The second formula maps the probability from the interval [0.5,1] back to the interval [0,1] and divides the result by the sum of the weights of all edges that leave the source node X. This guarantees that the sum of the weights of all the edges that leave a node will be equal to one.

In the probabilistic graph that was constructed, the weight of each edge is equal to the probability that a user is interested in the destination concept given that they are interested in the source concept, where we assume that the user is interested in only one of the destination concepts.

4 Measuring the Semantic Distance Between Word Forms

We will next show how to compute the semantic similarity between two arbitrary word form nodes in the graph. Our algorithm will return a number that is between zero and one. One will be returned when the two word forms are the same. Also, note that it is perfectly reasonable for two word forms to represent completely unrelated concepts and the semantic similarity between the word forms to be equal to zero. The semantic distance will be computed as a function of the average of the probability that the first word form is relevant given the second word form is relevant and the probability that the second word form is relevant given the first word form is relevant.

Consider two nodes n_1 and n_k in the probabilistic graph. We will show three different ways to compute the probability that n_k is relevant given that n_1 is relevant. In Section 5, we will compare the accuracy of the different approaches.

4.1 Multiplication Approach

A version of this approach was initially published in [43]. Consider a node sequence $n_1 \cdots n_k$ that forms a directed acyclic path in the graph. Let A_i be a random variable that represents the event that n_i is relevant for $i = 1$ to k. From probability theory, we have the following equation.

$$\begin{aligned} P(A_2 \cdots A_k | A_1) = \frac{P(A_1 \cdots A_k)}{P(A_1)} = \\ = \frac{P(A_1)P(A_2|A_1)P(A_3|A_1A_2)\cdots P(A_k|A_1\cdots A_{k-1})}{P(A_1)} = \\ = P(A_2|A_1)P(A_3|A_1A_2)\cdots P(A_k|A_1\cdots A_{k-1}) \end{aligned} \tag{23}$$

Next, we will simplify the formula by assuming some level of independence. Suppose that the event A_i only depends on the preceding event A_{i-1}. This is the same assumptions that is made in Bayesian networks. Given this assumption, we can rewrite the equation as follows.

$$P(A_2 \cdots A_k | A_1) = P(A_2|A_1)P(A_3|A_2)\cdots P(A_k|A_{k-1}) \tag{24}$$

The idea of this approach is that if n_k is relevant because n_1 is relevant and there is an acyclic directed path $n_1 \cdots n_k$ in the graph, then the nodes $n_2, \ldots, n_k$ must also be relevant. Next, we can use the above formula and compute the probability that n_k is relevant given that n_1 is relevant by simply multiplying the weights of the edges along the path. If there are multiple paths between n_1 and n_k in the graph, then we can add the conditional probability from each path. The result will be a probability because the weights of the edges are normalized. (Note that this is not the case in [43].) The formulas for computing the conditional probability are shown next.

$$P(A_k|A_1) = \sum_{Pt \text{ is acyclic path from node } n_1 \text{ to node } n_k} P(Pt) \tag{25}$$

$$P(Pt) = \prod_{(n_i,n_j) \text{ is an edge in the path } Pt} edgeWeight(n_i,n_j) \tag{26}$$

The *edgeWeight* function simply returns the weight of the edge. Note that the algorithm is not deterministic because there are different ways to select disjoint paths between two nodes in the graph. In our experiments we use the depth-first algorithm that is shown in Figure 1. Before calling the method, *totalDistance* is set to zero. After the method is called, the variable contains the result. When the method is initially called, *distance* is equal to one and *depth* is equal to zero. As the method is recursively called, the distance decreases and the depth is incremented by one after every call. In order to find the probability that n_k is relevant given that n_1 is relevant, we will call the method as follows: $depthFirst(n_1, n_k, 1, 0)$. The method starts at n_1 and recursively calls itself on all adjacent nodes in the graph. The recursion terminates when we have reached n_k, we have reached a node that has already been visited, we are on a path of more than 20 edges, or the value for the conditional probability for the path has dropped below the threshold of 0.0001.

4.2 Markov Logic Network Approach

A version of this approach was initially published in [46]. This approach is similar to the previous algorithm in the sense that the conditional probabilities over the different paths are aggregated. However, this approach uses the MLN approach to compute the conditional probability along a single path.

Let n_1 and n_k be two nodes in the probabilistic graph. We will next describe an efficient way of computing the probability that n_k is relevant given that n_1 is relevant using only the evidence along the path $n_1 \cdots n_k$. From probability theory, we have the following formula.

$$P(rel(n_k)|rel(n_1)) = \frac{P(rel(n_1) \wedge rel(n_k))}{P(rel(n_1))} \tag{27}$$

Algorithm 1 $depthFirst(currentNode, endNode, distance, depth)$

```
if $currentNode = endNode$ then
    $totalDistance \leftarrow totalDistance + distance$
    return
end if
if $depth > 20$ or $distance < 0.0001$ or $currentNode$ is visited then
    return
end if
for all neighbors $neighbor$ of $currentNode$ do
    $depthFirst(neighbor, endNode, distance * edgeWeigth(currentNode, neighbor), depth + 1)$
end for
```

Figure 1: Recursive method for finding disjoint paths between two nodes and computing the conditional probability

We will next show how to compute the numerator and denominator of the above expression using the weights of the edges along the path $n_1 \cdots n_k$.

Let $f00(i)$ be the non-normalized probability from Equation 22 (i.e., we do not divide by *total*) that n_i and n_k are both irrelevant. Similarly, let $f01(i)$ be the non-normalized probability that n_i is irrelevant and n_k is relevant, $f10(i)$ be the non-normalized probability that n_i is relevant and n_k is irrelevant, and $f11(i)$ be the non-normalized probability that both n_i and n_k are relevant. In order to understand why we need these functions, note that Equation 27 can be rewritten as follows.

$$\frac{P(rel(n_1) \wedge rel(n_k))}{P(rel(n_1))} = \frac{f11(1)}{f10(1) + f11(1)} \tag{28}$$

The numerator expresses the non-normalized probability that both n_1 and n_k are relevant. The non-normalized probability of n_1 being relevant is computed as $f10(1) + f11(1)$. The reason is that this formula computes the probability that n_1 is relevant and n_k is irrelevant plus the probability that n_1 is relevant and n_k is relevant, which is equal to exactly the probability that n_1 is relevant. Lastly, note that the fact that the probabilities are not-normalized will not affect the result because we divide a non-normalized probability by a non-normalized probability. That is, if the probabilities are normalized, then we will divide both the numerator and the denominator of the expression by the same constant *total* from Equation 22 and the result will not change.

We will compute $f00$, $f01$, $f10$, and $f11$ using dynamic programming. Using MLN theory, we have the following base case.

$$\begin{aligned}
f00(k-1) &= \frac{1 + edgeWeight(n_{k-1}, n_k)}{1 - edgeWeight(n_{k-1}, n_k)} \\
f01(k-1) &= \frac{1 + edgeWeight(n_{k-1}, n_k)}{1 - edgeWeight(n_{k-1}, n_k)} \\
f10(k-1) &= 1 \\
f11(k-1) &= \frac{1 + edgeWeight(n_{k-1}, n_k)}{1 - edgeWeight(n_{k-1}, n_k)}
\end{aligned} \tag{29}$$

The four values follow from Equation 22 and Equations 1 and 2. Note that we have the following formula and evidence probability.

$$rel(n_{k-1}) \Rightarrow rel(n_k), (edgeWeight(n_{k-1}, n_k))$$

The weight of the formula can be computed using Equations 1 and 2 as $ln(\frac{0.5 + \frac{edgeWeight(n_{k-1}, n_k)}{2}}{1 - (0.5 + \frac{edgeWeight(n_{k-1}, n_k)}{2})})$, which is equal to $ln(\frac{1 + edgeWeight(n_{k-1}, n_k)}{1 - edgeWeight(n_{k-1}, n_k)})$. Now, if n_{k-1} is not relevant and n_k is not relevant, then the formula $rel(n_{k-1}) \Rightarrow rel(n_k)$ will be true and according to Equation 22 the non-normalized probability for this world will be equal to $e^{ln(\frac{1 + edgeWeight(n_{k-1}, n_k)}{1 - edgeWeight(n_{k-1}, n_k)})} = \frac{1 + edgeWeight(n_{k-1}, n_k)}{1 - edgeWeight(n_{k-1}, n_k)}$. However, if n_{k-1} is relevant and n_k is irrelevant, then the formula will be false and the non-normalized probability will be equal to $e^0 = 1$.

Next, we present the recursive formulas for computing the four functions.

$$\begin{aligned}
f00(i) &= f00(i+1) * \frac{1 + edgeWeight(n_i, n_{i+1})}{1 - edgeWeight(n_i, n_{i+1})} + \\
&\quad f10(i+1) * \frac{1 + edgeWeight(n_i, n_{i+1})}{1 - edgeWeight(n_i, n_{i+1})} \\
f10(i) &= f00(i+1) * 1 + f10(i+1) * \frac{1 + edgeWeight(n_i, n_{i+1})}{1 - edgeWeight(n_i, n_{i+1})} \\
f01(i) &= f01(i+1) * \frac{1 + edgeWeight(n_i, n_{i+1})}{1 - edgeWeight(n_i, n_{i+1})} + \\
&\quad f11(i+1) * \frac{1 + edgeWeight(n_i, n_{i+1})}{1 - edgeWeight(n_i, n_{i+1})} \\
f11(i) &= f01(i+1) * 1 + f11(i+1) * \frac{1 + edgeWeight(n_i, n_{i+1})}{1 - edgeWeight(n_i, n_{i+1})}
\end{aligned} \tag{30}$$

Let us examine the first formula in detail. In this case, we want to compute the non-normalized probability of the world where both n_i and n_k are irrelevant. We have two sub-cases: when n_{i+1} is relevant and when n_{i+1} is irrelevant. When n_{i+1} is relevant, the following formula will be true.

$$rel(n_i) \Rightarrow rel(n_{n+1}), (edgeWeight(n_i, n_{i+1})) \tag{31}$$

We will therefore add to the probability $f00(i+1) * e^{ln(\frac{1 + edgeWeight(n_i, n_{i+1})}{1 - edgeWeight(n_i, n_{i+1})})}$ in this case, which is equal to $f00(i+1) * \frac{1 + edgeWeight(n_i, n_{i+1})}{1 - edgeWeight(n_i, n_{i+1})}$. We use the expression $f00(i+1)$ because we know that both n_{i+1} and n_k are irrelevant in this sub-case. The second sub-case is when n_{i+1} is irrelevant. The above formula will be true again and therefore we add to the probability the expression $f10(i+1) * \frac{1+edgeWeight(n_i,n_{i+1})}{1-edgeWeight(n_i,n_{i+1})}$.

Next, let us examine the second formula from Equation 30. In this case, we want to compute the non-normalized probability of the world where n_i is relevant, but n_k is irrelevant. We have two sub-cases: when n_{i+1} is relevant and when n_{i+1} is irrelevant. When n_{i+1} is irrelevant, Equation 31 does not hold and therefore will add the probability $f00(i+1) * e^0$. The second sub-case is when n_{i+1} is relevant and we will add the probability $f10(i+1) * \frac{1+edgeWeight(n_i,n_{i+1})}{1-edgeWeight(n_i,n_{i+1})}$ because Equation 31 holds. The last two formulas from Equation 30 can be derived similarly.

Note that our program for computing the f functions uses dynamic programming instead of recursion and runs in linear time relative to the size of the path. It first computes the value for the functions with input $k-1$ and then it applies the formulas from Equation 30 with values for i from $k-2$ up to 1. At the end, Equation 28 can be applied to find the conditional probability along the path $n_1 \cdots n_k$. If there are multiple paths along n_1 and n_k, then the conditional probabilities from the disjoint paths are aggregated using the algorithm from Figure 1.

4.3 Markov Logic Network Combined with Random Walk

Our experimental section (Section 5) shows that this approach produces the most accurate results. The drawback of the two previous approaches is that only disjoint paths between the nodes that are compared are explored. However, in most cases there are multiple interweaving paths between the two nodes and looking at only disjoint paths is not a very accurate approximation of the conditional probability. Here, we propose a simple alternative using a random walk. The algorithm from Figure 2 starts at *currentNode* and randomly visits 20 nodes in the search of *endNode*. If *endNode* is found, then the algorithm returns 1. Otherwise, it returns 0. We call this algorithm 10,000 times for the two nodes that we are comparing and aggregate the result. If we divide the total by 10,000, then we will get the conditional probability that the second node is relevant given that the first node is relevant, where the accuracy will be 4 digits after the decimal dot. We chose to look at paths of at most 20 nodes because we believe that longer paths give very little evidence about the semantic relationship between the word forms that the nodes represent.

Note that it is possible for the *randomWalk* algorithm to reach a dead end. For example, if we reach a node and there are no adjacent nodes that are not visited, the algorithm will return 0. This means that the random walk was unable to find the *endNode*. Specifically, the algorithm tries 100 times to find an adjacent node that is not visited and it gives up if it is unable to find such a node.

Algorithm 2 *randomWalk(currentNode, endNode)*

```
for $i \leftarrow 0$ to 20 do
    if $currentNode = endNode$ then
        return 1
    end if
    repeat
        $nextNode \leftarrow getRandomNextNode(currentNode)$
    until $nextNode$ is not already visited or loop has run for 100 times
    if above loop ran 100 times then
        return 0
    end if
    $curentNode \leftarrow nextNode$
end for
return 0
```

Figure 2: The method takes a random walk from *currentNode* and it returns 1 if it reaches *endNode* and 0 otherwise

4.4 Linear and Logarithmic Distance Metrics

Let $P(Y|X)$ denote the result of computing $P(rel(Y)|rel(X))$ using one of the three algorithms that we presented in the last three subsections. Next, we present two functions for measuring semantic similarity between two word forms. The linear function is shown in Equation 32.

$$|wf_1, wf_2|_{lin} = min(\alpha, \frac{P(wf_1|wf_2) + P(wf_2|wf_1)}{2}) * \frac{1}{\alpha} \quad (32)$$

The minimum function is used in order to cap the value of the similarity function at one. The coefficient α amplifies the available evidence ($\alpha \leq 1$). The experimental section of the article shows how the value for α is picked. Note that when α is equal to one, then the function simply takes the average of the two numbers and caps the result at one.

The second semantic similarity function is inverse logarithmic, that is, it amplifies the smaller values. It is shown in Equation 33. The *norm* function simply multiplies the result by a constant (i.e., $-log_2(\alpha)$) in order to move the result value in the range [0,1]. Note that the *norm* function does not affect the correlation results. Again, the experimental section of the article shows how the value for α is picked.

$$|wf_1, wf_2|_{log} = norm(\frac{-1}{log_2(min(\alpha, \frac{P(wf_1|wf_2)+P(wf_2|wf_1)}{2}))}) \quad (33)$$

5 Experimental Validation

The system consists of two programs: one that creates the probabilistic graph and one that queries the graph. We used the Java API for WordNet Searching (JAWS) to connect to WordNet. The interface was developed by Brett Spell [40]. All experiments were performed on a laptop with Intel i7 CPU and 16GB of main memory. It takes about three minutes to build the probabilistic graph and save it to the hard disk. The size of the graph file is 81MB and it easily fits in main memory. It takes about 5 seconds to load the graph in main memory. We will refer to the three algorithms for finding the conditional probability between two nodes as the Multiplication, MLN, and MLN+Random Walk. The average time for computing the similarity distance between two word forms is about 100 milliseconds for the first two algorithms and about 1 second for the MLN+Random Walk algorithm. It takes about three minutes to build the initial probabilistic graph.

We evaluated our system on five different benchmarks. For each benchmark, experiments with human subjects were conducted and the average human judgment for each pair of words was recorded. The *RG65 data set* was created by Rubenstein and Goodenough and contains 65 pairs of words ([33]). The *MC28 dataset* contains 28 pairs of words and was created by Miller and Charles [21]. The *Agirre201 dataset* contains 201 pairs of words and was developed by Agirre et al. [1]. It is a subset of the *WordSim-353 dataset* that contains 353 pairs or words and was created by Finkelstein et al. [8]. Pierro and Euzenant recently ran a new study on the RG65 dataset and got slightly different results ([24]) – we will refer to this benchmark as the *P&S$_{full}$ dataset*. Lastly, the *SimLex665 dataset* contains 665 pairs of words and was introduced by Hill et al. [12]. This happens to be the largest and most recent word similarity benchmark in literature.

For each dataset, we computed the Person and Spearman correlation between the data from the studies and the data that was produced by our system. The Person correlation is computed as shown in Equation 34. Note that we have used $\bar{X}$ to define the average of the numbers in the vector. We assume that the two vectors: $X = \langle x_1, \ldots, x_n \rangle$ and $Y = \langle y_1, \ldots, y_n \rangle$ are the input to the formula.

$$PearsonCorrelation(X,Y) = \frac{\sum_{i=1}^{n}(x_i - \bar{X})(y_i - \bar{Y})}{\sqrt{\sum_{i=1}^{n}(x_i - \bar{X})^2}\sqrt{\sum_{i=1}^{n}(y_i - \bar{Y})^2}} \tag{34}$$

A notable property of the Pearson correlation is that it is invariant as regards to any Euclidean operation, such as scaling, translation, or rotation of the data.

The formula for the Spearman correlation is shown in Equation 35.

$$SpearmanCorrelation(X,Y) = 1 - \frac{6\sum_{i=1}^{n}(rank(x_i) - rank(y_i))^2}{n(n^2-1)} \tag{35}$$

The $rank(x_i)$ expression returns the position of the number x_i in the sorted version of the list X. A notable property of the Spearman correlation is that it is rank invariant, that is, a monotonic transformation would not affect its value.

In Tables 1, 2, and 3 we show the Pearson, Spearman, and the average of the two correlations, respectively, for our algorithms. We compare our results to the current state-of-the-art, which includes 16 algorithms. The correlation data for these 16 algorithms was taken as report by Lastra-Diaz and Garcia-Serrano in [18].

Note that both our linear and logarithmic similarity metrics take as input the parameter α (see Equations 32 and 33). Table 4 shows the values for α that were used to create our experimental results. These values were selected because they produce the highest Pearson correlation for the RG65 dataset. It turns out that they are close to optimal (i.e., produce the highest correlation) for both the Pearson and Spearman correlation on the other benchmarks as well.

Looking at Table 1, we see that our MLN+RandomWalk algorithm that uses the logarithmic similarity metric gives us the highest Pearson correlation on three of the five benchmarks. More over, this algorithm also gives us the highest value for the average of the Pearson correlation over the five benchmarks. These are significant results that demonstrate the high quality of the data inside the probabilistic graph. It is also worth noting that the MLN+RandomWalk algorithm produces higher average Pearson correlation than the MLN algorithm, which in turns produces higher average correlation than the Multiplication algorithm. The reason is that the MLN+RandomWalk algorithm is based on strict probabilistic theory and is able to take into account the interweaving paths in the graph between the two nodes that we are comparing. Note as well that the logarithmic similarity metric produces slightly better results than the linear case. Specifically, the average over the three algorithms is 0.7513 for the logarithmic similarity metric and 0.7440 for the linear one.

Next, consider Table 2. Again, the MLN+RandomWalk algorithm that uses the logarithmic similarity metric produces Spearman correlation that is higher than the current state-of-the-art algorithms on three of the five benchmarks. In addition, the algorithm produces the highest value for the average of the Spearman correlation over the five benchmarks. Again, the logarithmic similarity metric produces a little higher correlation: 0.6931 average Spearman correlation for the linear case and 0.6996 average Spearman correlation for the logarithmic case. Note that the Spearman correlation for the MLN algorithm is the same for the linear and logarithmic similarity distance metric. The reason is that $\alpha = 0.3$ for both algorithms. This number means that results that are equal to above 0.3 for both metrics are mapped to 1. In other words, the ranking

Table 1: Pearson correlation on the five different benchmarks (the highest values are in bold)

Algorithm/Data Set	*RG65*	*MC28*	*Agirre201*	$P\&S_{full}$	*SimLex665*	*Average*
$Resnik_{ic-treebank-add1}$ [29]	0.8653	0.8809	0.6913	0.9003	0.5955	0.7867
Yuan et al. [52]	0.8675	0.8407	0.7061	0.9082	**0.6106**	0.7866
Seco et al. [36]	0.8642	0.8557	0.6969	0.9042	0.6048	0.7852
Sanchez et al. [34]	0.8752	0.8595	0.6946	0.9025	0.5941	0.7852
Meng et al. [19]	0.8723	0.8393	0.7039	0.9057	0.6010	0.7844
Harispe et al. [11]	0.8589	0.8575	0.6960	0.9003	0.6056	0.7836
$Resnik_{ic-semcorraw-add1}$ [29]	0.8658	0.8621	0.6955	0.8997	0.5930	0.7832
Sanchez et al.[35]	0.8616	0.8507	0.6973	0.9042	0.5995	0.7827
CondProbCosine [18]	0.8634	0.8562	0.6902	0.9015	0.5964	0.7815
CondProbHypo [18]	0.8658	0.8552	0.6874	0.9015	0.5940	0.7808
CondProbLeaves [18]	0.8635	0.8511	0.6891	0.9008	0.5934	0.7796
$CPCorpus_{ic-treebank-add1}$ [18]	0.8633	0.8678	0.6807	0.8987	0.5863	0.7794
$CPCorpus_{ic-semcorraw-add1}$ [18]	0.8647	0.8504	0.6792	0.8979	0.5843	0.7753
Zhou et al. [53]	0.8589	0.8403	0.6848	0.8905	0.5985	0.7746
$CondProbLogistic_{k8}$ [18]	0.8692	0.8142	0.6809	0.9064	0.5972	0.7736
Hadj Taieb et al. [50]	0.7933	0.6899	0.6490	0.8167	0.4921	0.6570
$Multiplication\,(linear)$	0.8690	0.8391	0.6256	0.8993	0.3995	0.7265
$Multiplication\,(log)$	0.8536	0.8220	0.5962	0.8996	0.4392	0.7221
$MLN\,(linear)$	0.8173	0.8653	0.7115	0.8475	0.4438	0.7371
$MLN\,(log)$	0.8160	0.8661	**0.7273**	0.8382	0.4575	0.7410
$MLN + RandomWalk\,(linear)$	0.8874	0.8913	0.7002	0.9152	0.4472	0.7683
$MLN + RandomWalk\,(log)$	**0.8992**	**0.9290**	0.7105	**0.9237**	0.4914	**0.7908**

Table 2: Spearman correlation on the five different benchmarks (the highest values are in bold)

Algorithm/Data Set	*RG65*	*MC28*	*Agirre201*	$P\&S_{full}$	*SimLex665*	*Average*
$Resnik_{ic-treebank-add1}$ [29]	0.7831	0.8882	0.6461	0.7783	0.5810	0.7353
Yuan et al. [52]	0.8206	0.8274	0.6656	0.8199	**0.6027**	0.7473
Seco et al. [36]	0.8012	0.8727	0.6643	0.7919	0.5901	0.7441
Sanchez et al. [34]	0.8034	0.8492	0.6576	0.8003	0.5906	0.7402
Meng et al. [19]	0.8166	0.8296	0.6581	0.8127	0.5957	0.7426
Harispe et al. [11]	0.7977	0.8697	0.6539	0.7904	0.5918	0.7407
$Resnik_{ic-semcorraw-add1}$ [29]	0.7922	0.8712	0.6505	0.7835	0.5782	0.7351
Sanchez et al.[35]	0.7911	0.8551	0.6590	0.7854	0.5850	0.7351
CondProbCosine [18]	0.7896	0.8606	0.6524	0.7834	0.5828	0.7337
CondProbHypo [18]	0.8017	0.8554	0.6466	0.7910	0.5806	0.7350
CondProbLeaves [18]	0.7877	0.8389	0.6478	0.7808	0.5799	0.7270
$CPCorpus_{ic-treebank-add1}$ [18]	0.7722	0.8502	0.6364	0.7691	0.5735	0.7203
$CPCorpus_{ic-semcorraw-add1}$ [18]	0.7916	0.8247	0.6389	0.7813	0.5712	0.7216
Zhou et al. [53]	0.8051	0.8244	0.6591	0.7999	0.5945	0.7366
$CondProbLogistic_{k8}$ [18]	0.7993	0.8034	0.6460	0.7921	0.5791	0.7240
Hadj Taieb et al. [50]	0.7417	0.6961	0.6175	0.7463	0.4833	0.6570
$Multiplication\,(linear)$	0.7365	0.7653	0.4893	0.7348	0.3964	0.6245
$Multiplication\,(log)$	0.7424	0.7859	0.4969	0.7456	0.4140	0.6370
$MLN\,(linear)$	0.7704	0.8420	**0.6953**	0.7687	0.4552	0.7063
$MLN\,(log)$	0.7704	0.8420	**0.6953**	0.7687	0.4552	0.7063
$MLN + RandomWalk\,(linear)$	0.8375	0.9236	0.6789	0.8235	0.4794	0.7486
$MLN + RandomWalk\,(log)$	**0.8392**	**0.9423**	0.6801	**0.8253**	0.4909	**0.7556**

Table 3: Average of Pearson and Spearman correlation on the five benchmarks (the highest values are in bold)

Algorithm/Data Set	*RG65*	*MC28*	*Agirre201*	*P&S$_{full}$*	*SimLex665*	*Average*
Resnik$_{ic-treebank-add1}$ [29]	0.8242	0.8846	0.6687	0.8393	0.5883	0.7610
Yuan et al. [52]	0.8441	0.8341	0.6859	0.8641	**0.6067**	0.7670
Seco et al. [36]	0.8327	0.8642	0.6806	0.8481	0.5975	0.7647
Sanchez et al. [34]	0.8393	0.8544	0.6761	0.8514	0.5924	0.7627
Meng et al. [19]	0.8445	0.8345	0.6810	0.8592	0.5984	0.7635
Harispe et al. [11]	0.8283	0.8636	0.6750	0.8454	0.5987	0.7622
Resnik$_{ic-semcorraw-add1}$ [29]	0.8290	0.8667	0.6730	0.8416	0.5856	0.7592
Sanchez et al.[35]	0.8264	0.8529	0.6782	0.8448	0.5923	0.7589
CondProbCosine [18]	0.8265	0.8584	0.6713	0.8425	0.5896	0.7576
CondProbHypo [18]	0.8338	0.8553	0.6670	0.8463	0.5873	0.7579
CondProbLeaves [18]	0.8256	0.8450	0.6685	0.8408	0.5867	0.7533
CPCorpus$_{ic-treebank-add1}$ [18]	0.8178	0.8590	0.6586	0.8339	0.5799	0.7499
CPCorpus$_{ic-semcorraw-add1}$ [18]	0.8282	0.8376	0.6591	0.8396	0.5778	0.7485
Zhou et al. [53]	0.8320	0.8324	0.6720	0.8452	0.5965	0.7556
CondProbLogistic$_{k8}$ [18]	0.8343	0.8088	0.6635	0.8493	0.5882	0.7488
Hadj Taieb et al. [50]	0.7675	0.6930	0.6333	0.7815	0.4877	0.6570
$Multiplication\,(linear)$	0.8028	0.8022	0.5575	0.8171	0.3980	0.6755
$Multiplication\,(log)$	0.7980	0.8040	0.5466	0.8226	0.4266	0.6795
$MLN\,(linear)$	0.7939	0.8537	0.7034	0.8081	0.4495	0.7217
$MLN\,(log)$	0.7932	0.8541	**0.7113**	0.8035	0.4564	0.7237
$MLN + RandomWalk\,(linear)$	0.8625	0.9075	0.6896	0.8694	0.4633	0.7584
$MLN + RandomWalk\,(log)$	**0.8692**	**0.9357**	0.6953	**0.8745**	0.4912	**0.7732**

Table 4: Values for α

Algorithm	α *linear metric*	α *log metric*
Multiplication	0.002	0.1
MLN	0.3	0.3
MLN+RandomWalk	0.006	0.015

is the same after applying the linear or logarithmic similarity distance metric and therefore the Spearman correlation is the same. Lastly, note that again the MLN+RandomWalk algorithm produces the highest average correlation, followed by the MLN and the Multiplication algorithm. However, the MLN algorithm produces the best results on the Agirre201 benchmark.

Lastly, consider Table 3 that shows the average of the Pearson and Spearman correlation. Again, the MLN+RandomWalk algorithm that uses the logarithmic similarity measure produces higher correlation than previous algorithms on four of the five benchmarks and the highest average correlation over the five benchmarks. The logarithmic similarity metric produces a little higher correlation than the linear one: 0.7185 average correlation for the linear case and 0.7255 average correlation for the logarithmic case. The MLN+RandomWalk algorithm produces higher average correlation than the MLN algorithm, which produces higher average correlation than the Multiplication algorithm.

The Java source code and all text files that are needed to reproduce the experimental results can be found at [41].

6 Conclusion and Future Research

In this article, we presented a new Markov Logic Network algorithm that uses a random walk to compute the semantic similarity between two word forms of the English language. We showed that the logarithmic version of the algorithm produces higher average correlation over five benchmarks than the current state-of-the-art algorithms. We believe that these results are due to the fact that our algorithm processes not only structured data, but also natural language information from WordNet. Moreover, unlike our previous work, the algorithm considers all the evidence from the probabilistic graph and not only the disjoint paths between the nodes that are compared.

Although the random walk algorithm gives very accurate results, it is not necessarily the most efficient way of computing the semantic similarity between two nodes in the probabilistic graph. In the future, we plan to explore alternative methods for computing the semantic similarity distance between two nodes, such as Gibbs sampling, belief propagation, and approximation via pseudolikelihood. We also plan on conducting experiments on the full-blown version of the probabilistic graph that includes data from Wikipedia ([42]) and determining if this can improve the correlation values with the five benchmarks.

References

[1] E. Agirre, E. Alfonseca, K. Hall, J. Kravalova, M. Pasca, and A. Soroa. "A Study on Similarity and Relatedness using Distributional and WordNet-based Approaches."

Proceedings of Human Language Technologies: The 2009 Annual Conference of the North American Chapter of the Association for Computational Linguistics, pp. 19–27, 2009.

[2] R. Blanco, P. Mika, and S. Vigna. “Effective and Efficient Entity Search in RDF Data.” Tenth International Conference on The Semantic Web, pp. 83–97, 2011.

[3] D. Bollegala, Y. Matsuo, and M. Ishizuka. “A Relational Model of Semantic Similarity Between Words Using Automatically Extracted Lexical Pattern Clusters from Web.” Conference on Empirical Methods in Natural Language Processing, 2009.

[4] L. Burnard. “ Reference Guide for the British National Corpus (XML Edition).” http://www.natcorp.ox.ac.uk, 2007.

[5] P. Castells, M. Fernandez, and D. Vallet. “An Adaptation of the Vector-space Model for Ontology-based Information retrieval.” IEEE Transactions on Knowledge and Data Engineering, 19(2):pp. 261–272, 2007.

[6] R. L. Cilibrasi and P. M. Vitanyi. “The Google Similarity Distance.” IEEE ITSOC Inforamtion Theory Workshop, 2005.

[7] S. Deerwester, S. T. Dumais, G. W. Furnas, T. K. Landauer, and R. Harshman. “Indexing by Latent Semantic Analysis.” Journal of the Society for Information Science, 41(6):pp. 391–407, 1990.

[8] L. Finkelstein, E. Gabrilovich, Y. Matias, E. Rivlin, Z. Solan, G. Wolfman, and E. Ruppin. “Placing Search in Context: The Concept Revisited.” ACM Transactions on Information Systems, 20(1):pp. 116–131, January 2002.

[9] C. Fox. “Lexical Analysis and Stoplists.” Information Retrieval: Data Structures and Algorithms, pp. 102–130, 1992.

[10] W. Frakes. “Stemming Algorithms.” Information Retrieval: Data Structures and Algorithms, pp. 131–160, 1992.

[11] S. Harispe, S. Ranwez, S. Janaqi, and J. Montmain. “Semantic Similarity from Natural Language and Ontology Analysis.” Synthesis Lectures on Human Language Technologies, pp. 81–254, 2015.

[12] F. Hill, R. Reichart, and A. Korhonen. “SimLex-999: Evaluating Semantic Models with (Genuine) Similarity Estimation.” arXiv:1408.3456, 2014.

[13] G. Jeh and J. Widom. “SimRank: A Measure of Structural-context Similarity.” Proceedings of the Eight ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 538–543, 2002.

[14] K. Jones. “A Statistical Interpretation of Term Specificity and its Application in Retrieval.” Journal of Documentation, 28(1):pp. 11–21, 1972.

[15] R. Knappe, H. Bulskov, and T. Andreasen. “Similarity Graphs.” Fourteenth International Symposium on Foundations of Intelligent Systems, 2003.

[16] S. Kulkami and D. Caragea. “Computation of the Semantic Relatedness Between Words Using Concept Clouds.” International Conference of Knowledge Discovery and Information Retrieval, 2009.

[17] T. K. Landauer, P. Foltz, and D. Laham. “Introduction to Latent Semantic Analysis.” Discourse Processes, pp. 259–284, 1998.

[18] J. J. Lastra-Diaz and A. Garcia-Serrano. “A New Family of Information Content Models with an Experimental Survey on WordNet.” Elsevier Knowledge-Based Systems, 89(1):pp. 509–526, 2015.

[19] L. Meng, J. Gu, and Z. Zhou. “A New Model of Information Content Based on Concept’s Topology for Measuring Semantic Similarity in WordNet.” International Journal on Grid Distributed Computing, 5(3):pp. 81–93, 2012.

[20] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean. “Distributed Representations of Words and Phrases and their Compositionality.” Advances in Neural Information Processing Systems, 26(1):pp. 3111–3119, 2013.

[21] G. Miller and W. Charles. “Contextual Correlates of Semantic Similarity.” Language and Cognitive Processing, 6(1):pp. 1–28, 1991.

[22] G. A. Miller. “WordNet: A Lexical Database for English.” Communications of the ACM, 38(11):pp. 39–41, 1995.

[23] R. Pan, Z. Ding, Y. Yu, and Y. Peng. “A Bayesian Network Approach to Ontology Mapping.” Proceedings of the Fourth International Semantic Web Conference, pp. 563–577, 2005.

[24] G. Pirro and J. Euzenat. “A Feature and Information Theoretic Framework for Semantic Similarity and Relatedness.” Proceedings of the ninth international semantic web conference on the semantic web, pp. 615–630, 2010.

[25] M. F. Porter. “An Algorithm for Suffix Stripping.” Readings in Information Retrieval, pp. 313–316, 1997.

[26] Q. Rajput and S. Haider. “Use of Bayesian Networks in Information Extraction from Unstructured Data Sources.” Proceedings of International Conference on Ontological and Semantic Engineering, pp. 325–331, 2009.

[27] RDF Wordking Group. “Resource Description Framework (RDF).” http://www.w3.org/RDF/, 2014.

[28] P. Resnik. “Using Information Content to Evaluate Semantic Similarity in a Taxonomy.” International Joint Conference on Artificial Intelligence, pp. 448–453, 1995.

[29] P. Resnik. “Semantic Similarity in a Taxonomy: An Information-based measure and its application to problems of ambiguity in natural language.” Journal of Artificial Intelligence, 11:pp. 95–130, 1999.

[30] M. Richardson and P. Domingos. “Markov Logic Networks.” Machine Learning, 62(1-2):pp. 107–136, 2006.

[31] S. Robertson and H. Zaragoza. “The Probabilistic Relevance Framework: BM25 and Beyond, Foundations and Trends in Information Retrieval.” Foundations and Trends in Information Retrieval, 3(4):pp. 333–389, 2009.

[32] C. Rocha, D. Schwabe, and M. Aragao. “A Hybrid Approach for Searching in the Semantic Web.” Thirteenth International World Wide Web Conference (WWW 2004), pp. 374–383, 2004.

[33] H. Rubenstein and J. B. Goodenough. “Contextual Correlates of Synonymy.” Communications of the ACM, 8(10):pp. 627–633, 1965.

[34] D. Sanchez, M. Baret, and D. Isern. “Ontology-based Information Content Computation.” Knowledge-Based Systems, 24(2):pp. 297–303, 2011.

[35] D. Sanchez and M. Batet. “A New Model to Compute the Information Content of Concepts from Taxonomic Knowledge.” International Journal on Semantic Web Information Systems, 8(2):pp. 34–50, 2012.

[36] N. Seco, T. Veale, and J. Hayes. “An Intrinsic Information Content Metric for Semantic Similarity in WordNet.” Sixteenth European Conference on Artificial Intelligence, 16:pp. 1089–1094, 2014.

[37] Simone Paolo Ponzetto and Michael Strube. “Deriving a Large Scale Taxonomy from Wikipedia.” 22nd International Conference on Artificial Intelligence, 2007.

[38] S. Simske, I. Boyko, and G. Koutrika. “Multi-Engine Search and Language Translation.” ExploreDB, 2014.

[39] E. Sirin and B. Parsia. “SPARQL-DL: SPARQL Query for OWL-DL.” 3rd OWL: Experiences and Directions Workshop (OWLED), 2007.

[40] B. Spell. “Java API for WordNet Searching (JAWS).” http://lyle.smu.edu/ tspell/jaws/index.html, 2009.

[41] L. Stanchev. “Experimental Results.” http://users.csc.calpoly.edu/∼lstanche/kbs.

[42] L. Stanchev. “Creating a Phrase Similarity Graph from Wikipedia.” Eight IEEE International Conference on Semantic Computing, 2014.

[43] L. Stanchev. “Creating a Similarity Graph from WordNet.” Fourth International Conference on Web Intelligence, Mining and Semantics, 2014.

[44] L. Stanchev. “Fine-Tuning an Algorithm for Semantic Search Using a Similarity Graph.” International Journal on Semantic Computing, 9(3):pp. 283–306, 2015.

[45] L. Stanchev. “Semantic Search using a Similarity Graph.” Ninth IEEE International Conference on Semantic Computing, 2015.

[46] L. Stanchev. “Creating a Probabilistic Graph for WordNet using Markov Logic Network.” Sixth International Conference on Web Intelligence, Mining and Semantics, 2016.

[47] L. Stanchev. “Semantic Document Clustering Using a Similarity Graph.” Tenth IEEE International Conference on Semantic Computing, 2016.

[48] M. Steyvers and J. Tenenbaum. “The Large-Scale Structure of Semantic Networks: Statistical Analyses and a Model of Semantic Growth.” Cognitive Science, 29(1):pp. 41–78, 2005.

[49] N. Stojanovic. “On Analyzing Query Ambiguity for Query Refinement: The Librarian Agent Approach.” Twenty Second International Conference on Conceptual Modeling, pp. 490–505, 2003.

[50] M. A. H. Taieb, M. B. Aouicha, and A. B. Hamadou. “Ontology-based Approach for Measuring Semantic Similarity.” Elsevier Engineering Applications of Artificial Intelligence, 36:pp. 238–261, 2014.

[51] The World Wide Web Consortium. “OWL Web Ontology Language Guide.” http://www.w3.org/TR/owl-guide/, 2014.

[52] Q. Yuan, Z. Yu, and K. Want. “A New Model of Information Content for Measuring the Semantic Similarity Between Concepts.” International Conference on Cloud Computing and Big Data, pp. 141–146, 2013.

[53] Z. Zhou, Y. Want, and J. Gu. “A New Model of Information Content for Semantic Similarity in WordNet.” Second International Conference on Future Generation Communication and Networking Symposia, 3:pp. 85–89, 2008.

Lubomir Stanchev is an Associate Professor at California Polytechnic State University, San Luis Obispo, California. Before joining Cal Poly, he was an Assistant and then an Associate Professor at Indiana-University Purdue-University Fort Wayne. He got his Ph.D. in Computer Science from the University of Waterloo in Canada. His research interests include databases, parallel algorithms, and semantic computing. He has published more than thirty papers in peer-reviewed conferences and journals.

Virtual Watershed Visualization for the WC-WAVE Project

Chase Carthen, Thomas J. Rushton, Nolan P. Burfield,
Christine M. Johnson, Aaron Hesson, Daniel Nielson and Bryan Worrell
University of Nevada at Reno, Reno, NV 89557 USA

Donna Delparte, Tucker Chapman and W. Joel Johansen
Idaho State University, Pocatello, ID 83209 USA

Roger Lew, Nicholas R. Wood, Mathew Ziegler and John W. Anderson
University of Idaho, Moscow, ID 83844 USA

Sergiu M. Dascalu and Frederick C. Harris Jr.
University of Nevada at Reno, Reno, NV 89557 USA
Fred.Harris@cse.unr.edu

Abstract

The platform discussed in this paper, Virtual Watershed Client, provides a tool that allows researchers, students, and stakeholders to observe and analyse both geospatial datasets and theoretical model data. The system created ingests the geospatial data and can create three-dimensional terrain, satellite imagery, shapes, and model data. This was done using the Unity3D game development engine and libraries to work with geospatial data such as GDAL and NetCDF. These tools that are developed are used by researchers, most specifically hydrologist for the purpose of this paper, to interact in a three-dimensional environment with their gathered data. The design of the Virtual Watershed Client, as well as its currently implemented analysis and visualization tools are discussed in detail throughout this paper.

Key Words: Visualization, geospatial, model data, hydrology, virtual watershed platform, virtual watershed client, terrain.

1 Introduction

Today there is a need to visualize and analyze datasets captured by sensors in an environmental watershed and produced by biophysical models. This need stems from the watershed scientists; the ability to view their collected and model data on an actual terrain is beneficial. Having the ability to see the peaks and valleys of data values line up in accordance to actual peaks and valleys of the terrain is just one example of the usefulness visualizing the datasets. Researchers use sensors to provide information about the surrounding environment of the watershed and they allow the researchers to run models to analyze the data captured from these sensors. Researchers use this acquired data to develop models that effectively reflect the environment of the watershed. These models can then be used to predict water runoff, precipitation, snow melt, underground water flow, and other environmental factors within a watershed. Models consist of many input and output variables that require visualization and tuning. There are several existing tools used to visualize this data and provide a researcher sufficient information to modify variables in their models. These tools include QGIS [30], ArcMap [6], and other Geographical Information System (GIS) and non-GIS programs. However, most of these tools lack the ability to visualize and interact with data from a first person perspective, that is to say viewing the watershed as if you were actually there in person. The Virtual Watershed Client [4] is currently being implemented and has been designed to provide researchers a 3D analytical tool to visualize model data. Researchers will be able to provide data to a web service, the Virtual Watershed Platform, and the Virtual Watershed Client can then be used to visualise this data. The Virtual Watershed Platform uses the Open Geospatial Consortium (OGC) services for serving data [28]. The client uses libraries such as the Geospatial Data Abstraction Library (GDAL) to parse any data that gets delivered to the Virtual Watershed Client [10]. Researchers are then able to observe this visualization to evaluate the outcomes of their model runs. With the modern gaming development engines, and the processing power of computers fusing a game atmosphere with large data manipulation is possible. The current implementation of the Virtual Watershed Client was built using the Unity game engine [36]. The Unity Game Engine was chosen because it allows our platform to be published on many potential platforms, such as Windows, Mac, Linux, and the web browser.

The capabilities of researchers to gather data in their respective watershed environments has coincided with the ability to aggregate and analyze that data in meaningful ways. The authors of [29] discuss methods of determining the optimal locations for dam placement in order to harvest water. They do this by analyzing a multitude of data maps overlaid onto a Digital Elevation Model (DEM). In [23] the authors describe a toolkit built for hydrological modeling using a geographic information system such as ArcView GIS. Their system, the Automated Geospatial Watershed Assessment tool (AGWA) is capable of executing and then visualizing results inside of ArcView GIS from models such as the Soil and Water Assessment Tool (SWAT) and Kinematic Runoff and Erosion Model (KINEROS2). The virtual watershed implemented in the Sevier River Basin [3] was designed for the purpose of improving operation of river and irrigation canals. To do this, a real-time representation of the watershed was created using remote sensors that constantly stream data to a server, allowing for rapid decision making. Finally, in Advanced Techniques for Watershed Visualization [1], the authors discuss using hydroshading algorithms on NASA's Shuttle Radar Topography Mission DEM datasets in order to generate high quality 3D models of the watersheds that the the datasets pertain to.

The following discussion expands upon the details given in the paper Design of a Virtual Watershed Client for the project [4]. Section 2 discusses the overview of the funded project, and how the Virtual Watershed Client fits into that project. Section 3 explains the models used by the watershed scientists on the funded project. Section 4 covers the details of the design and implementation of the Virtual Watershed Client. Next, we demonstrate two separate watersheds in Section 5, those two watersheds being located in Dry Creek and Lehman Creek. Finally, we present conclusions and future work of this project in Section 6.

2 Project Overview

The Western Consortium for Watershed Analysis, Visualization, and Exploration (WC-WAVE) [38] is a tri-state consortium composed of researchers and students in Idaho, Nevada, and New Mexico. Its overall aim is to study the localized impact of climate change on high-mountain catchments. The WC-WAVE project is comprised of three components: Watershed Science, Visualization and Data Cyberinfrastructure (CI), and Workforce Development. Participants in these three components are collaborating to better understand interactions between precipitation, snow-pack, groundwater flow, and other watershed properties within mountain catchments. To enhance the collaborative nature of this project and future work, as well as to promote data exploration and analysis, one of the main outcomes of the WC-WAVE project is a Virtual Watershed Platform (VWP). The VWP framework will provide data access and visualization through individual workstations (desktop), web-based environments, and advanced interactive 3D environments, including stereoscopic, and immersive CAVE and Virtual Reality environments. The VWP will also allow a user to run specific hydrologic models and visualize the results. During the preliminary meetings for this proposal, project team members put together the design for the Virtual Watershed Platform. Figure 1 illustrates this VWP design.

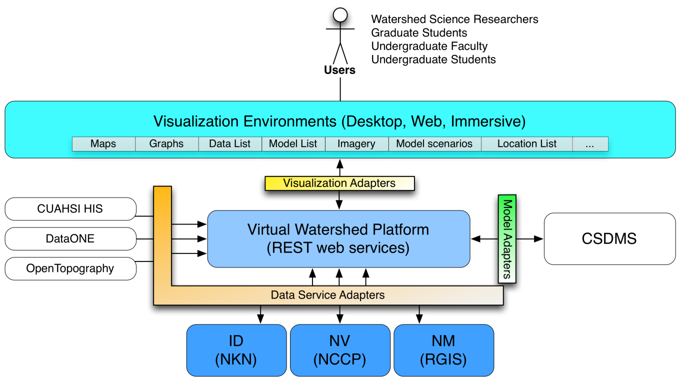


Figure 1: Diagram illustrating the relationships of proposed project components; virtual watershed platform, data management services, and portals. The Virtual Watershed Client is on the Visualization Environments block, which gets data required from the Virtual Watershed Platform. The data to the VWP comes from multiple locations. Then all data transfer is done through adapters

To implement and test the VWP, project participants selected four watersheds throughout the tri-state region to focus on: Dry Creek, Reynolds Creek Experimental Watershed in Idaho, Jemez watershed in New Mexico, and Lehman Creek in Nevada. These watersheds were chosen partly for their unique characteristics. For example, Dry Creek and Reynolds Creek provide an ideal environment for studying and developing tools related to snow pack, while the Jemez watershed is more suitable for vegetation studies. These watersheds were also chosen for their extensive data networks. Since the models within the VWP call for several different forms of data, watersheds with adequate datasets and instrumentation were required.

Members of the Watershed Science component of the WC-WAVE project selected four main models to include in the VWP: ISNOBAL [18], PRMS [35], ParFlow [20], and DFLOW [34]. The ISNOBAL (image SNOw mass and energy BALance) model is used to predict snowmelt and runoff. PRMS (Precipitation Runoff Modeling System) simulates the hydrologic process. ParFlow (PARallel FLOW) model is useful for modeling soil moisture and the interaction between surface and sub-surface flow. The DFLOW model is used for simulating river channel flow and hydrologics. These datasets are later explained in Section 3.

3 Data Models

As stated, the WC-WAVE project uses four primary models for conducting watershed science. Out of the four models ISNOBAL is the most established in VWC. During the development of the VWC, ISNOBAL datasets were most readily available for testing and development purposes, therefore features were designed around the data provided from this model. Section 3.1 describes this model in depth, and discusses one of the watersheds in the project that contains ISNOBAL data. The other models used in VWC are represented in the same method as ISNOBAL, and are described in Section 3.2.

3.1 Snowpack Modeling

Several models have been developed for the purpose of predicting snowpack properties over the course of a season. SNOW17 [2] was designed as part of the National Weather Service's River Forecasting System, and is still in use today [8]. The SNTHERM [13] and SHAW [7] models accurately simulate snow properties at a point, while the Utah Energy Balance (UEB) model [33] and the United States Geological Survey's Precipitation-Runoff Modeling System [15] can be applied as distributed simulations over small areas. SnowModel [16] utilizes input meteorological conditions along with wind conditions, vegetation and surface energy exchange to simulate snow depth and water equivalent. The ISNOBAL model [19] is part of our current implementation, as well as the Image Processing Workbench (IPW) suite of software tools [17, 37].

ISNOBAL is effective at predicting snowmelt and runoff when applied to a range of watershed sizes (1 to 2500 km2) as well as temporal ranges (one week to an entire season). When run through IPW, ISNOBAL requires three different types of input: an initial conditions image, precipitation images, and input forcing data images. The same initial condition image can be used for an entire model run and is comprised of seven bands: elevation, roughness length, total snowcover depth, average snowcover density, active snow layer temperature, average snowcover temperature, and percentage of liquid H_2O saturation. Precipitation images consist of four bands: total precipitation mass, percentage of precipitation mass that was snow, density of snow portion of precipitation, and average precipitation temperature. These bands are only included for times of precipitation events (i.e. storms). Individual input forcing data images are required for each time step (typically three hours) throughout the model run, and are made up of six bands: incoming thermal (long-wave) radiation, air temperature, vapor pressure, wind speed, soil temperature, and net solar (short-wave) radiation. All of these images can be visualized in the 3D environment of the VWP. This allows the user to check for errors, anomalies, and patterns before an actual model run. Using these images as input, ISNOBAL uses a two layer snow model to calculate energy and mass balance terms and produces two images as output. The first is a ten-band energy and mass flux image which includes predicted evaporation, snowmelt, and runoff layers. The second is a nine-band snow conditions image which contains layers for predicted thickness of snowcover, snow density, and mass of the snowcover. Similar to the input images, these output layers can be visualized with the VWP for further analysis. A visualization of several of the ISNOBAL variables can be seen in Figure 2.

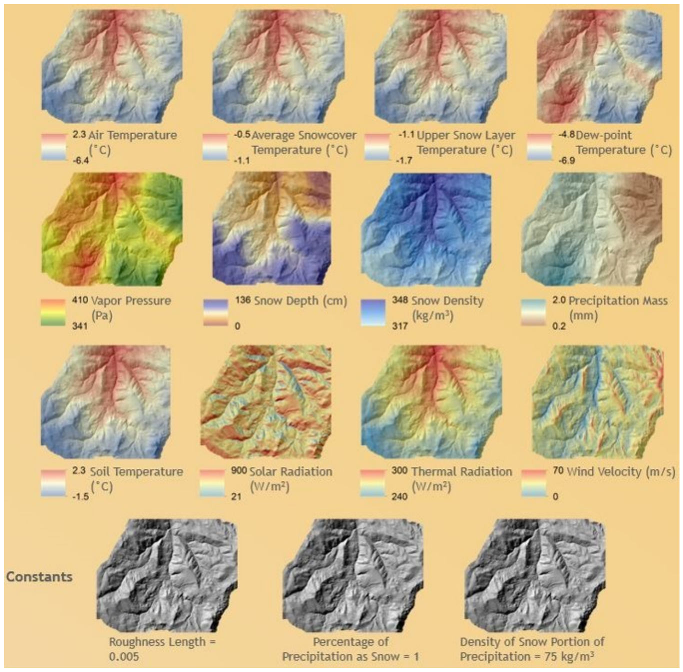

2.3 Air Temperature (°C) -6.4
-0.5 Average Snowcover Temperature (°C) -1.1
-1.1 Upper Snow Layer Temperature (°C) -1.7
-4.8 Dew-point Temperature (°C) -6.9
410 Vapor Pressure (Pa) 341
136 Snow Depth (cm) 0
348 Snow Density (kg/m³) 317
2.0 Precipitation Mass (mm) 0.2
2.3 Soil Temperature (°C) -1.5
900 Solar Radiation (W/m²) 21
300 Thermal Radiation (W/m²) 240
70 Wind Velocity (m/s) 0
Constants
Roughness Length = 0.005
Percentage of Precipitation as Snow = 1
Density of Snow Portion of Precipitation = 75 kg/m³

Figure 2: A visualization of different ISNOBAL Variables

3.2 Other Data Models

3.2.1 The DFLOW Model calculates the flows of streams specifically for low flowing streams. DFLOW takes daily stream flow information, and can calculate biologically-based design flows, hydrologically-based design flows, and harmonic and percentile flows. This model can be used to calculate across multiple bodies of water.

3.2.2 The PRMS Model is part of our current implementation. The Precipitation Runoff Modeling System (PRMS) has been developed to simulate the hydraulic processes, simulate the hydraulic water budgets of the watershed, integrate with other models, and have a modular design to allow alternative hydrologic-process algorithms. The hydrologic processes modelled in the system include evaporation, transpiration, runoff, infiltration, and interflow. The interflow is determined by the waterbudgets of plant canopy, snowpack, and soil zones calculated by temperature, precipitation, and solar radiation. Additionally, the hydraulic water budget simulation produces temporal data ranging in scale from days to years [35].

3.2.3 The ParFlow Model (PARallel FLOW) simulates surface and subsurface flow as an integrated hydrological model. This model runs the parallel simulation with an option of three modes. The first two simulation modes, steady-state saturated and variably saturated, show the flow through heterogeneous porous media. The third mode enables the coupling of the surface and subsurface flow. This enables ParFlow to account for the hillslope runoff and channel routing. ParFlow is the massively parallel computations that run advanced numerical solvers and multigrid preconditioners. This parallel environment takes advantage of octrees to run octree-space partitioning in order to handle input variables in three-space. These features allow ParFlow to run large scale watershed simulations [20].

4 Design and Implementation of the Virtual Watershed

The primary intention for the Virtual Watershed Client is to aid in the visualization of data collected by environmental scientists for better understanding of the watershed environment and the models (ISNOBAL, Parflow, etc.). Several dependencies are needed for the Virtual Watershed Client to be functional. The Virtual Watershed Client requires the Unity game engine, libraries similar to GDAL, and the OGC services. These dependencies in conjunction with the Unity game engine make it possible to implement a geospatial tool and application that is unique in comparison to other applications. The Virtual Watershed Clients architecture, functional requirements, implementation, and main use cases are explained in detail in this section.

4.1 Model View Controller Architecture

The Virtual Watershed Client utilizes a Model-View-Controller (MVC) architectural pattern for the user interface (Figure 3). The Model component is responsible for querying the Virtual Watershed Platform for data, the View component is responsible for displaying the data with the currently applied settings, and the Controller is responsible for handling any configuration changes and movement in the world generated by the VWC. The Model component retrieves data through the OGC services that can be displayed in the form of terrains, shapes (rivers, streams, and roads), model data, and imagery. The model and view component will be discussed more in this section.

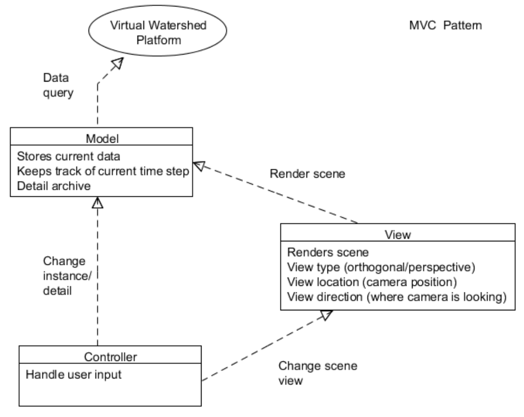

Virtual Watershed Platform
MVC Pattern
Data query
Model
Stores current data
Keeps track of current time step
Detail archive
Render scene
View
Renders scene
View type (orthogonal/perspective)
View location (camera position)
View direction (where camera is looking)
Change instance/ detail
Controller
Handle user input
Change scene view

Figure 3: A representation of the Model View Controller architecture of the Virtual Watershed Client

4.2 Use Cases

The users of the Virtual Watershed Client are able to create watershed simulations using data retrieved from the Virtual Watershed Platform. In addition to selecting the data for the simulation, users also have the option of configuring various settings to analyze desired variables of the data in more detail. One of the approaches taken to enable this level of analysis is to allow the user to immerse themselves into the simulation in the sense that he or she can interact with various objects throughout the environment such as trees and sensors. Users also have the option to configure settings for data download speeds, image quality, etc., in order to give greater flexibility in viewing aspects of the data that are most important to the specific users analyses. Once data has been downloaded, users will also be able to configure the representation of that data on the terrain. Figure 4 demonstrates a general use case model [32].

The system being referred to in Figure 4 is the running application of the Virtual Watershed Client [14]. Within the use case model, the server is any outside service that the system utilizes to acquire data. When the application launches, the user can select a dataset to be loaded from a file or downloaded from the server, as well as configure the watershed. The system will use the selected dataset and configurations to start and generate a simulation. Then the Virtual Watershed Client will display the terrain and user/model selected weather conditions. Once the visualization has begun, the user can configure a model run, and have it prepared for simulation by the system. Any data required for the generation of the model run will be acquired from the server at this time. While the model is running, information corresponding to water runoff, snow melt, underground moisture, etc. will be displayed.

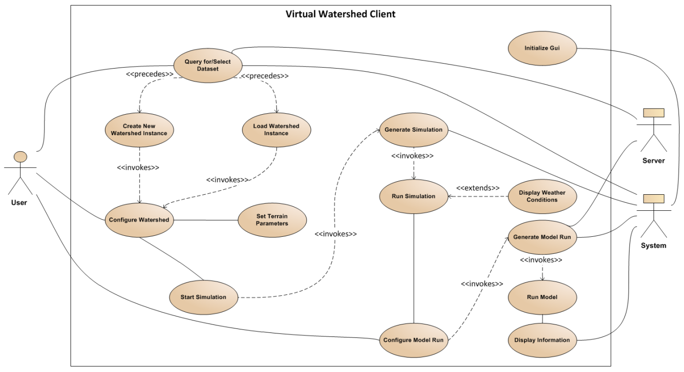


Figure 4: A use case model of the Virtual Watershed Client demonstrates different scenarios from the perspectives of a user, the VWC, and the data server of the VWP

The use cases of Figure 4 are detailed below:

Initialise GUI

The GUI of the Virtual Watershed Client is initialized.

Query for/ Select Dataset

Datasets can be queried for or selected that are acquired from the virtual watershed platform or cached by the VWP.

Create New Watershed Instance
The user has the option of creating a new watershed instance by selecting a different watershed to start in.

Configure Watershed
The VWC can be configured where the user can change what datasets are added to the current watershed.

Start Simulation
Once the watershed has been properly configured the watershed can start loading all necessary datasets and load a scene.

Generate Simulation
Once the simulation is started the scene in Unity will be generated.

Run Simulation
After the scene has been generated, the scene will continue to run automatically.

Set Terrain Parameters
Several options are available for the user to select options such as where the terrain will be located in the real world.

Configure Model Run
The Virtual Watershed Client allows the user to configure a model run for running it.

Generate Model Run
After a model run has been configured it can be generated.

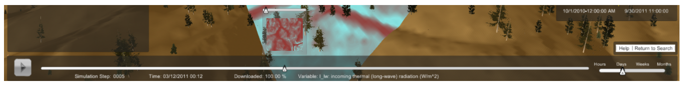
Display Weather Conditions
The virtual watershed has the ability to visualize different weather conditions that the user can select.

Run Model
The user can run models from VWC using the VWP. Currently the VWC has the ability visualize datasets from previously ran models.

Display Information
All of the current datasets, location, elevation, and other useful conditions are provided for the user to view either through the UI or by inspecting the dataset itself.

4.3 Graphical User Interface

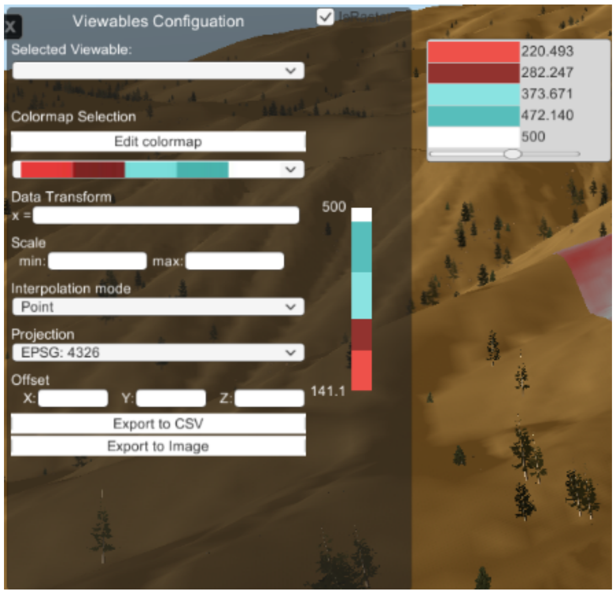
4.3.1 The Time Slider shown in Figure 5 on the bottom of the image is a tool that allows users to view data maps over time. Once a model run has been downloaded and selected for viewing by the user, it will be loaded into the time slider, as well as a projector onto the terrain in the data's correct geospatial positioning. There are multiple components of the Time Slider feature. The bar through the center of the Time Slider contains a cursor that shows the current data's position in time related to the overall dataset. The text fields underneath this bar display the information such as the count of the frame currently being displayed, the time that the data corresponds to, how much of the dataset has been downloaded into the Time Slider, and the model run variable that the data represents. On the right side of the Time Slider is a scale configuration tool. This slider allows the user to specify the speed at which they wish to view the data, depending on the overall time differential between the individual data maps. The play button will simply play through the data at the time specified on the speed slider. Finally, the window above the cursor of the Time Slider shows a replica of the data being displayed via the projector. In this way, even if the user is not able to view the data directly on the terrain from their current position, they can see the representation of data played out through time.



Simulation Step: 0005
Time: 03/12/2011 00:12
Downloaded: 100.00 %
Variable: I_lw: incoming thermal (long-wave) radiation (W/m^2)

Figure 5: The time slider used to visualize temporal data sets such as model run variables

4.3.2 The Configuration Panel to the left of Figure 6 is used to configure the representation of data in the virtual world. One of the major ways that the data can be configured, is by editing the color-map that is applied to the data. When the data is loaded, the minimum and maximum values are recorded and set to specific color values. All potential values of the data points between the minimum and maximum are then mapped to an interpolated color defined by the five colors of the color-map. These color ranges are predefined at first, but the user can change the value ranges that each color represents. Multiple predefined color-maps are available to the user, with options available to accommodate users with color-blindness. Extending from the color-map is the ability to project the resulting data using either a point or bi-linear interpolation mode. The point interpolation mode simply matches one data point in the grid to a single color value, whereas the bilinear mode blends between the values of the surrounding points to produce a more natural looking representation of the data.



Selected Viewable:
Colormap Selection
Data Transform
x =
Scale
min: max:
Interpolation mode: Point
Projection: EPSG: 4326
Offset
X: Y: Z:

🟥	220.493
🟥	282.247
🟦	373.671
🟦	472.140
⬜	500

Figure 6: The configuration panel used to alter aspects of data presentation on the terrain

At the bottom of the panel are three text fields, allowing the user to shift the data projection or models in the X, Y, and Z coordinates. This is necessary for cases that a dataset did not come with a predefined position, or that the position given was incorrect. Finally, there are options to export the current data frame being displayed to either an image, showing the color-map representation, or a comma separated value (CSV) file with the values of each point in the data frames grid.

4.3.3 The Data-Point Graph shown in the bottom right of the Figure 7 displays the value at a point on the dataset throughout the time-line of the dataset. The user is able to select a single location on the projected data and the graph will display the data values at that spatial location plotted over the time range of the dataset. The vertical yellow bar of the graph shows the user where in time the currently projected data frame is in relation to the graph as a whole. If the user has two datasets loaded, then there will be two lines present on the graph. This feature allows the user to compare data values of a specific location over two separate model runs or variables. The user will also have the option to download the current graph data as a CSV.

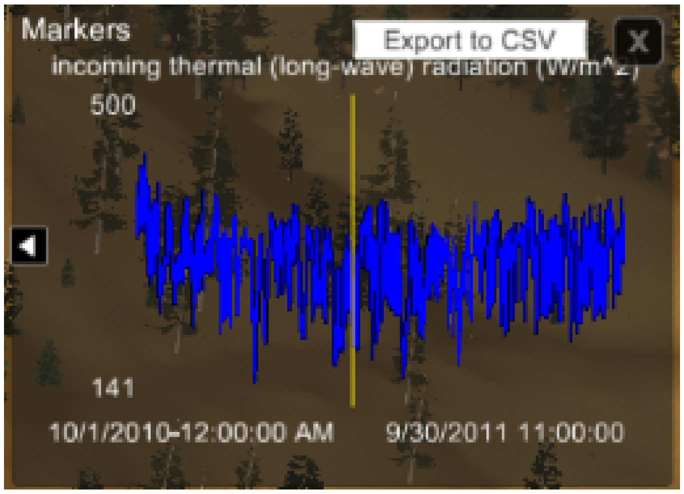


Figure 7: The data-point graph showing the fluctuating data values of a point over time

4.3.4 The Terrain Slicer on the right side of Figure 8 is used for taking 2D cross-sections of the terrain. Two slicer nodes can be placed in the world, and the window of the Terrain Slicer will show a 2D image representing the elevation of the terrain between the two nodes placed. This feature allows for exciting future developments, including the ability to take slices of not only the terrain, but 3D datasets. For example, ParFlow data that models underground water flow could be displayed using the slicer to visualize water flow or ground moisture at differing depths.

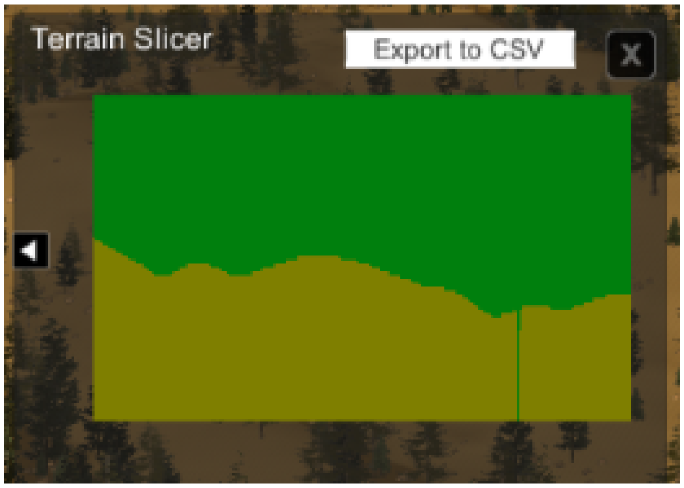


Figure 8: The terrain slicer showing a 2D slice of the environment height-map

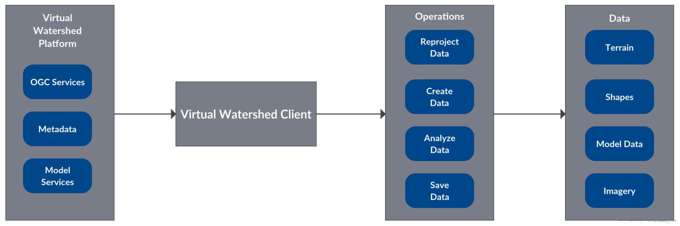


Figure 9: A programmatic flow diagram of the Virtual Watershed Client. The VWP box indicates some of the different services that the VWC can communicate with and the operations box indicates the different operations that the VWC can take on the data. The data box demonstrates the core types of data that the VWC receives from the VWP

4.4 Implementation

4.4.1 Virtual Watershed Client The VWC has several dependencies that make it possible to bring geospatial datasets into Unity for visualization and for analysis. Unity was chosen to be used as the visualization due its cross platform capabilities and its relative ease of use in developing visualizations that could take longer in lower level graphics pipelines such as OpenGL or DirectX. Unity actually handles the lower level graphics pipelines in its own implementation. These dependencies are GDAL, ProjNet, JSON.net, and Gavaghan Geodesy library. GDAL can parse several different geospatial data formats such as geotiffs, netcdf, and many other formats. Both GDAL and ProjNet include the core functionality to do different types of geospatial calculations such as to reproject one coordinate system into another. JSON.net was included in the VWC due to its capability to parse JSON files and has been primarily for parsing the RESTful services coming from the VWP. Figure 9 demonstrates the functionality of the VWC in a block diagram.

The VWC makes use of the RESTful services from the VWP to provide the user with data to generate a watershed and visualize different model runs stored on the VWP. One key thing that the VWC has to do is be able to internalize these datasets into a common form that the application can then generate the proper visualization. Several different parsers have been written using the dependencies described previously to bring these datasets into the VWC. As the datasets are acquired from the Virtual Watershed they will be cached onto local file system to allow for fast loading if the dataset is selected again for loading. These parsers allow us to extract data to build visualizations of many different datasets such as terrains, models that were explained previously, shape files, and images such as landsat imagery. These visualizations are made possible by the API and functionality provided by Unity. Another key feature of the VWC is the ability to incorporate other data sets granted that another parser is written for that particular server.

4.4.2 Data Capabilities

The Virtual Watershed Client has the capability to load datasets from the VWP or from the local file system. The VWP serves many different types of data in various formats such as NetCDF [31], TIFF, shapefile, JSON, XML, CSV, and other formats supported by GDAL due to the OGC services supported by the VWP. We use GDAL to parse these formats whether they are from a web service inside of the VWP, or from the local file system into a common data format that is understandable to the VWC. GDAL effectively expands the VWC and allows us to support multiple formats. Along with GDAL we have designed parsers for the VWP that allows us to parse and pull data out of the VWP itself.

The VWC has been designed in such a way that other web services can be easily incorporated as long as the perspective parsers are written for the web service. These web services can be other servers such as NASA Modis's [26] web services. With the incorporation of web services into the VWC, the ability to add other datasets and models is now possible.

With the ability to acquire datasets both locally and the VWP, the VWC is able to create different types of visualizations. These visualizations include procedurally built terrain, bring shape files that may be weather stations or roads, load imagery, and temporal-spatial data sets at run time rather than being preprocessed. With the ability to acquire datasets from web services, it will be possible to create other different types of visualizations such as procedurally vegetation, running models inside the VWP, and uploading datasets.

4.4.3 Coordinate System Design

The VWC supports a variety of coordinate systems such that it can represent world datasets in a 3D world and takes into consideration the distance in meters between two objects in order to properly map an object into the virtual world. Our coordinate system utilizes libraries such as GDAL and Gavaghan's Geodesy Library [9] to perform operations such as handling the placement of entities into the world, calculating distances between two points, and changing the projection of other datasets into the VWC's coordinate system. These libraries have allowed us to create a robust coordinate system that can handle a variety of datasets.

In our coordinate system we have decided to use the Universal Transverse Mercator (UTM) coordinate system projections to preserve euclidean relationships between two points. The UTM coordinate system allows the VWC to represent objects with accurate relational distances. Figure 10 demonstrates the UTM coordinate system's 60 longitudinal zones that the Earth is sliced into; measurements using UTM coordinates are usually done within a single zone. A GPS will make use of this coordinate system for relatively short distances due to its accuracy. We utilize this coordinate system for its accuracy in short distances allowing the VWC to place two objects within a reasonable distance of their real world positions, or to project imagery accurately. Though effective when working within an individual zone, the accuracy of the UTM coordinate system diminishes when taking into consideration points that span across two zones.

The limitations of the UTM coordinate system make it difficult to use datasets that span two zones. Calculating the distance between points overlapping a UTM zone boundary results in major distortions. Additionally, calculations of distance overlapping the equator produce similar distortions. These distortions make it necessary for the addition of other coordinate systems into the VWC that deal with these distortions and effectively estimate the distance for two points.

The coordinate system makes use of a frame of reference for placing real world objects into the virtual world of the Unity Game Engine. This frame of reference is paired with the origin in the Unity Game Engine and a location in the real world. This frame of reference allows the VWC to translate real world coordinates into the VWC's coordinate system. Real world points are coordinates having some geospatial relationship to the Earth. These coordinates can be either Latitude and Longitude, UTM points, or other types of coordinates. Unity Game Engine coordinates refer to points that are inside Unity Game Engine with one unit in the Unity Game Engine represented as one meter. This frame of reference makes it easy to convert

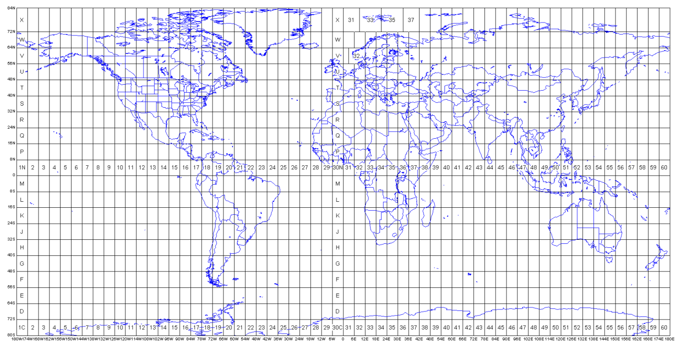


Figure 10: A cartographic projection of the Earth demonstrating the UTM Latitudinal and Longitudinal zones. Typically UTM uses the Longitudinal zones where each has a width of six degrees [25]

between the real world and the VWC's visualization. The frame of reference requires coordinates to be in the Latitude and Longitude system. We utilize Latitude and Longitude points since they are optimal in distance equations, and are easily converted to other coordinate systems. Our coordinate system first converts real world points into a Latitude and Longitude coordinate representation given that they were not originally presented in that format. Algorithm 1 demonstrates how the VWC can convert from real world points to the Unity coordinate system and Algorithm 2 demonstrates the inverse.

The two algorithms utilize the reference coordinate points in order to move between the Earth's coordinate system and Unity's coordinate system. The VWC uses two formula's (Haversine and Vincenty) to calculate the distance between two points. The implementation for Haversine comes from [5], and the implementation for Vincenty comes from [9]. Algorithm 1 is able to handle cases where a data point may be outside the zone currently being observed by the VWC. To account for these cases, Algorithm 3 is used to estimate the equivalent point in Unity. This algorithm estimates a projected point using either the Haversine or Vincenty formula based on the converted Latitude and Longitude calculated in Algorithm 1, and the world Latitude Longitude reference point. Despite the effectiveness of the Haversine and Vincenty formula, they are only accurate to a certain degree. Both of these formula's will work relatively well, however they may not accurately represent the scale of the UTM coordinate system. To account for these inaccuracies our system will allow the user to move datasets if their positions in the virtual world differ from the proper location.

Algorithm 1 World to Unity

```
1: function WORLD TO UNITY($WorldPoint$)
2:     $ConvertedWorldPoint \leftarrow$ CONVERTTOLATLONG($WorldPoint$)
3:     $Zone =$ CALCULATEZONE($ConvertedWorldPoint$)
4:     if $Zone \neq WorldZone$ then
5:         $ConvertedWorldPoint \leftarrow$ ESTIMATEPROJECTION($ConvertedWorldPoint$)
6:     else
7:         $ConvertedWorldPoint \leftarrow$ CONVERTTOUTM($ConvertedWorldPoint$)
8:     end if
9:     return $(UnityReferenceOrigin +$
10:        $(UTMWorldReferencePoint - ConvertedWorldPoint))$
11: end function
```

Algorithm 2 Unity to World

```
1: function UNITY TO WORLD($UnityPoint$)
2:     $ConvertedWorldPoint \leftarrow$ CONVERTTOLATLONG($UTMWorldReferencePoint + (UnityReferenceOrigin - UnityPoint), ZoneNumber$)
3:     return $ConvertedWorldPoint$
4: end function
```

Algorithm 3 Estimate Projection

```
1: function ESTIMATEPROJECTION($WorldPoint$)
2:     if $Vincenty$ then
3:         $Distance \leftarrow$
4:             VINCENTYDISTANCEESTIMATION($WorldPoint, LatLongWorldReferencePoint$)
5:     else if $Haversine$ then
6:         $Distance \leftarrow$
7:             HAVERSINEDISTANCEESTIMATION($WorldPoint, LatLongWorldReferencePoint$)
8:     end if
9:     $Direction \leftarrow WorldPoint - LatLongWorldReferencePoint$
10:    $ProjectedWorldPoint \leftarrow Distance * Direction$
11:    return $ProjectedWorldPoint$
12: end function
```

5 Virtual Watershed Examples: Lehman Creek and Dry Creek

The Unity engine is a platform for building interactive virtual environments. It provides an integrated development environment (IDE) consisting of a GUI and scripting to define the appearance and behavior of objects in the virtual environment. The Unity engine allows developers to combine 2D and 3D visual assets, audio, and network access to data. The Unity engine is built with Mono, which is a cross-platform open source implementation of the .NET framework [24, 22]. Using Mono allows for procedural control, creation, and manipulation of objects in UnityScript (Javascript) and C#. The engine provides a well optimized code base capable of displaying high-fidelity interactive 3D virtual environments with real-time physics and shading. Although Unity offers great potential to geoscience visualization and education outreach, the IDE has a steep learning curve and is not natively equipped with essential features like geo-referencing, Gregorian time, or the ability to handle geospatial datasets (GeoTiffs, ShapeFiles, etc.) common to GIS Desktop solutions.

The Virtual Watershed Client will provide a simple GUI for communicating to a centralized Virtual Watershed Platform. Acquiring datasets for visualization is now possible through the VWP within the VWC or the local file system. Several visualizations have been developed for the Virtual Watershed Client using preprocessed files from the local file system. Datasets acquired from the VWP can be used to procedurally build a terrain for visualization at run time of the Virtual Watershed Client. We have constructed preprocessed terrains for the WC-WAVE project that are being used for analyzing datasets that are specific to the WC-WAVE project.

The process of building preprocessed terrains requires certain datasets to create a well built visualization. When available, high-resolution LiDAR can be represented by the

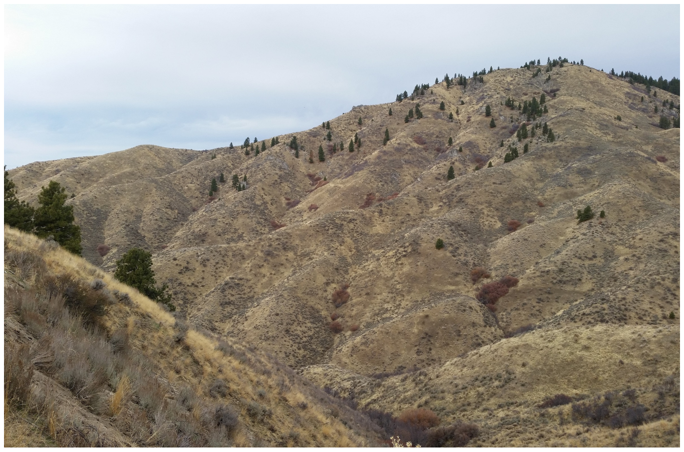


Figure 11: A photograph of Dry Creek

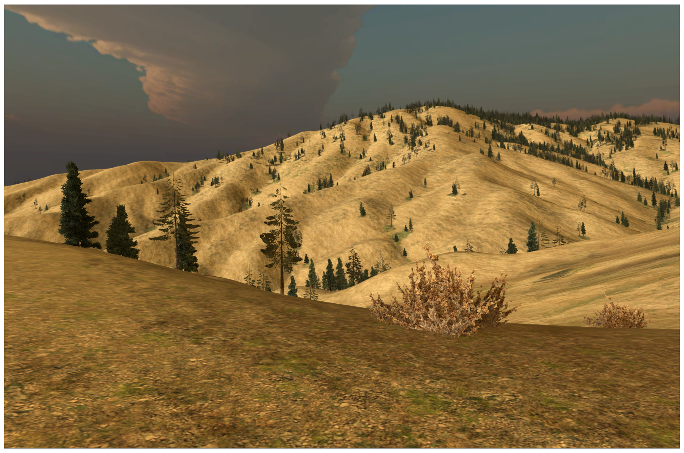


Figure 12: The VWC image of Dry Creek

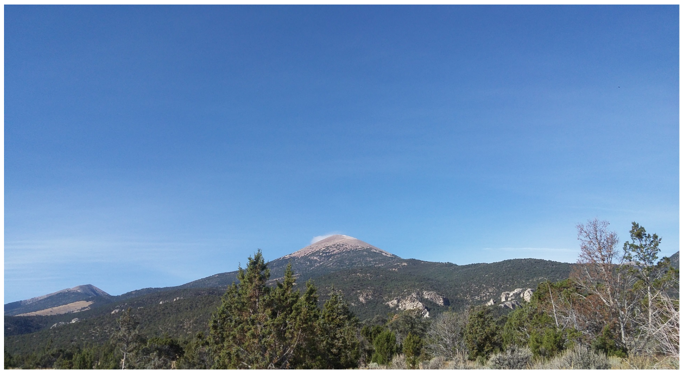


Figure 13: A photograph of Lehman Creek

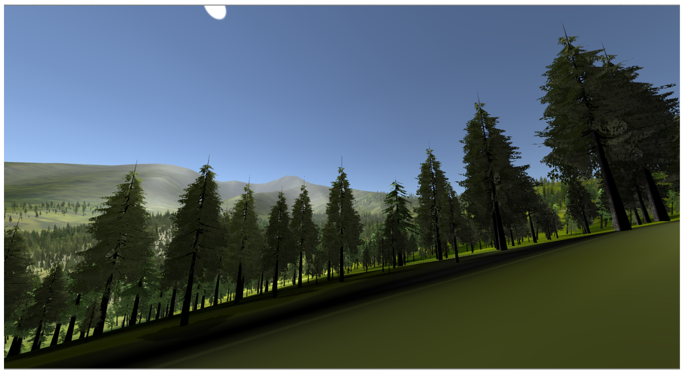


Figure 14: The VWC image of Lehman Creek

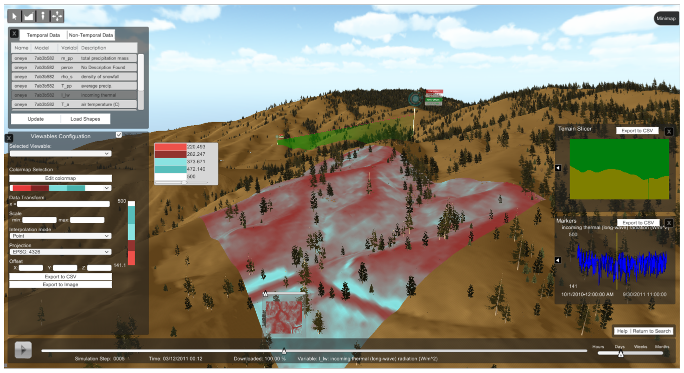


Figure 15: The entire user interface shown on the Dry Creek terrain with a thermal long wave radiation dataset projected on to the surface

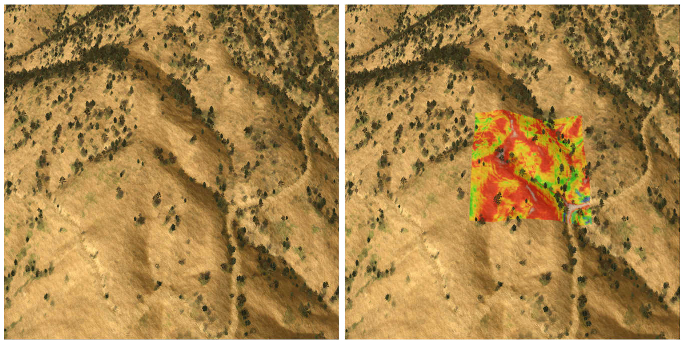


Figure 16: A picture of the Dry Creek sub-catchment with choropleth (right) and without choropleth (left)

terrain. High-resolution orthoimagery is then placed on the terrain, and procedurally splatted with detailed textures and vegetation (as shown in Figure 12 and Figure 14). Both Figure 12 and Figure 14 can be compared to their respective counterparts, Figure 11 and Figure 13, that demonstrate the overall effectiveness of our visualization.

Once a base map has been constructed, scientific data can be incorporated. A projector with a customized shader allows for choropleths (thematic maps) to be overlaid onto the terrain as shown in Figures 15 and 16. These maps could represent characteristics of terrain topology like slope or spatial-temporal variables associated with hydrology models such as ISNOBAL, Parflow, or PRMS. An example of a spatial-temporal output of an ISNOBAL variable can be seen in Figure 15.

We can also visualize the outputs of a PRMS model by playing back the output file onto the terrain as a choropleth. As explained previously in Section 4, the user can alter the values of the projector, change the times on the time slider, or compare two model runs.

6 Future Work and Conclusions

We have covered the design and implementation of the Virtual Watershed Client, the different models of the WC-WAVE project, and new additions to the VWC. This paper discussed the unique design of a three-dimensional project for the use with geospatial data. The project incorporates the GDAL libraries into a game development engine, Unity3D, for purposes of research. Additionally the unique tools designed into the project to interact with the data will help researchers gather more information from the provided data. The VWC has several new features that include elements that have been added to the UI, a new coordinate system, the ability to visualize spatial-temporal datasets, generate terrains procedurally, and the incorporation of the Virtual Watershed Platform. With the addition of these new features researchers can visit different areas in the world and utilize the new features for the purposes of analyzing datasets that are both stored locally and on the VWP.

In terms of future work the team is planning new features that would be both beneficial and needed for the Virtual Watershed Client. Integration with the VWP has allowed us to incorporate procedural terrain generation and allows us to consider implementing procedural vegetation generation. Being able to procedurally generate vegetation alongside terrain would allow us to simulate different models such as CASiMiR that have a vegetation output. Watershed researchers could effectively analyze the effect of changing seasons, climate, and fire scenarios. We will soon incorporate the ability to visualize vector motion fields that would be useful for showing stream flow in the bottom of a river or to demonstrate wind. To provide a more immersive experience of the Virtual Watershed Client with virtual reality, we would like to allow the user to to use Head Mounted Displays (HMD) such as the Oculus Rift [27], HTC Vive [12], or Google Cardboard [11]. We would like to also build a visualization for a six-sided CAVE (Cave auto virtual environment). We would like to incorporate other sensors such as the Microsoft Kinect [21] and others. Beyond new visualizations features and the incorporation of other sensors we have plans to incorporate other services from the Virtual Watershed Platform and other servers.

The Virtual Watershed Client currently is able to interface with the Virtual Watershed Platform and download datasets from it. We would like to expand our functionality to allow users to upload datasets to the Virtual Watershed Platform and run models on those datasets. While the Virtual Watershed Platform support for many other different types of datasets relate to the WC-WAVE project, we would want to include other servers such as NASA's Modis web services [26] that have global datasets that could be beneficial to the Virtual Watershed Client. Incorporation of these other web services would be both beneficial and useful for the Virtual Watershed Client and would allow for a greater diversity of datasets to be visualized and analyzed within the VWC.

Acknowledgments

This material is based in part upon work supported by the National Science Foundation under grant numbers: IIA-1329469, IIA-1329470, IIA-1329513, and IIA-1301726, and the EPSCoR MILES grant number: IIA-1301792. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation

References

[1] V. Alarcon and C. O'Hara, "Advanced Techniques for Watershed Visualization," *Applied Image Pattern Recognition Workshop*, vol. 35, 2006.

[2] E. A. Anderson, "National Weather Service River Forecast System Snow Accumulation and Ablation Model: NOAA Tech," *Memorandum NWS Hydro-17, US Dept. of Commerce*, pp. 3–7, 1973.

[3] B. Berger, R. Hansen, and I. Cowley, "Developing a Virtual Watershed: Sevier River Basin," Decision Support Systems for Water Resources Management (Specialty Conference). American Water Resources Association, 2001.

[4] C. D. Carthen, T. J. Rushton, C. M. Johnson, A. Hesson, D. Nielson, B. Worrell, D. M. Delparte, W. J. Johansen, J. W. Anderson, R. Lew, N. R. Wood, M. Ziegler, S. M. Dascalu, and F. C. Harris, "Design of a Virtual Watershed Client for the WC-WAVE Project," IEEE International Conference on Collaboration Technologies and Systems (CSA), June 2015, pp. 90–96.

[5] D. Dennehy, "Haversine Algorithm in C#," [Accessed on 15 March 2016]. [Online]. Available: http://damien.dennehy.me/blog/2011/01/15/haversine-algorithm-in-csharp/

[6] Esri, "Arcgis Platform," [Accessed on 14 March 2016]. [Online]. Available: www.esri.com/software/arcgis

[7] G. N. Flerchinger and K. E. Saxton, "Simultaneous Heat and Water Model of a Freezing Snow-Residue-Soil System I. Theory and Development," *Transactions of the ASAE*, vol. 32, no. 2, pp. 565–0571, 1989.

[8] K. J. Franz, T. S. Hogue, and S. Sorooshian, "Operational Snow Modeling: Addressing the Challenges of an Energy Balance Model for National Weather Service Forecasts," *Journal of Hydrology*, vol. 360, no. 1, pp. 48–66, 2008.

[9] M. Gavaghan, "C# Geodesy Library for GPS Vincenty's Formula," [Accessed on 1 March 2016]. [Online]. Available: http://www.gavaghan.org/blog/free-source-code/geodesy-library-vincentys-formula/

[10] Gdal, "Gdal - Geospatial Data Abstraction Library," [Accessed on 14 March 2016]. [Online]. Available: http://www.gdal.org/

[11] Google, "Google Cardboard," [Accessed on 29 February 2016]. [Online]. Available: www.vr.google.com/cardboard/

[12] HTC, "Htc vive," [Accessed on 29 February 2016]. [Online]. Available: www.htcvive.com/us/

[13] R. Jordan, "A One-Dimensional Temperature Model for a Snow Cover: Technical Documentation for SNTHERM. 89." DTIC Document, Tech. Rep., 1991.

[14] S. Lauesen, "Task Descriptions as Functional Requirements," *IEEE Software*, vol. 20, no. 2, pp. 58–65, 2003.

[15] G. H. Leavesley, R. W. Lichty, B. M. Thoutman, and L. G. Saindon, *Precipitation-Runoff Modeling System: User's Manual*. US Geological Survey Colorado, CO, 1983.

[16] G. E. Liston and K. Elder, "A Distributed Snow-Evolution Modeling System (SnowModel)," *Journal of Hydrometeorology*, vol. 7, no. 6, pp. 1259–1276, 2006.

[17] D. Marks, J. Domingo, and J. Frew, "Software Tools for Hydro-Climatic Modeling and Analysis: Image Processing Workbench, ARS-USGS Version 2," *ARS Tech. Bull. 99*, vol. 1, 1999.

[18] D. Marks, J. Domingo, D. Susong, T. Link, and D. Garen, "A Spatially Distributed Energy Balance Snowmelt Model for Application in Mountain Basins," *Hydrological Processes*, vol. 13, no. 12-13, pp. 1935–1959, 1999.

[19] ——, "A Spatially Distributed Energy Balance Snowmelt Model for Application in Mountain Basins," *Hydrological Processes*, vol. 13, no. 12-13, pp. 1935–1959, 1999.

[20] R. M. Maxwell, S. J. Kollet, S. G. Smith, C. S. Woodward, R. D. Falgout, I. M. Ferguson, C. Baldwin, W. J. Bosl, R. Hornung, and S. Ashby, "ParFlow User's Manual," *International Ground Water Modeling Center Report GWMI*, vol. 1, no. 2009, p. 129, 2009.

[21] Microsoft, "Kinect for Windows," [Accessed on 2 March 2016]. [Online]. Available: http://www.microsoft.com/en-us/kinectforwindows/

[22] ——, ".NET Framework and .NET SDK Downloads," [Accessed on 3 March 2016]. [Online]. Available: https://msdn.microsoft.com/en-us/vstudio/aa496123.aspx

[23] S. Miller, D. Semmens, D. Goodrich, M. Hernandez, R. Miller, W. Kepner, and D. Guertin, "The Automated Geospatial Watershed Assessment Tool," *Environmental Modelling & Software*, vol. 22, no. 3, pp. 365–377, 2007.

[24] Mono-Project, "Home — mono," [Accessed on 28 February 2016]. [Online]. Available: http://www.mono-project.com/

[25] A. Morton, "Dmap: UTM Grid Zones of the World," [Accessed on 1 March 2016]. [Online]. Available: http://www.dmap.co.uk/utmworld.htm

[26] National Aeronautics and Space Administration, "Modis Web," [Accessed on 3 March 2016]. [Online]. Available: http://modis.gsfc.nasa.gov

[27] Oculus, "Oculus Rift - Virtual Reality Headset for Immersive 3D Gaming," [Accessed on 29 February 2016]. [Online]. Available: www.oculus.com/

[28] Opengeospatial, "Open Geospatial Consortium — OGC," [Accessed on 14 March 2016]. [Online]. Available: http://www.opengeospatial.org/

[29] D. Patel, M. Dholakia, N. Naresh, and P. Srivastava, "Water Harvesting Structure Positioning by Using Geo-Visualization Concept and Prioritization of Mini-Watersheds Through Morphometric Analysis in the Lower Tapi Basin," *J Indian Soc Remote Sens*, vol. 40, no. 2, pp. 299–312, 2011.

[30] QGIS, "Qgis Project," [Accessed on 14 March 2016]. [Online]. Available: http://www2.qgis.org/en/site/

[31] R. Rew and G. Davis, "NetCDF: An Interface for Scientific Data Access," *Computer Graphics and Applications*, vol. 10, no. 4, pp. 76–82, 1990.

[32] I. Sommerville, *Software Engineering*. Boston: Pearson, 2011.

[33] D. G. Tarboton and C. H. Luce, *Utah Energy Balance Snow Accumulation and Melt Model (UEB)*. Citeseer, 1996.

[34] United States Environmental Protection Agency, "DFLOW — Water Data and Tools," [Accessed on 2 March 2016]. [Online]. Available: http://www.epa.gov/waterdata/dflow

[35] United States Geological Survey, "Prms," [Accessed on 11 March 2016]. [Online]. Available: ftp://brrftp.cr.usgs.gov/pub/mows/software/prms/release_notes.pdf

[36] Unity3D, "Unity - Game Engine," [Accessed on 14 March 2016]. [Online]. Available: http://unity3d.com/

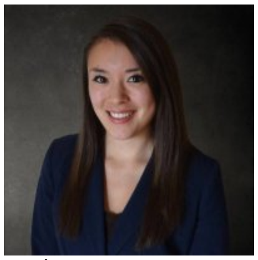
[37] USDA Agricultural Research Service, "Ipw Command: isnobal," [Accessed on 14 March 2016]. [Online]. Available: ftp://ftp.nwrc.ars.usda.gov/ipw/Marks%20et%20al%201999/man1/isnobal.html

[38] Western Consortium, "Western Tri-State Consortium," [Accessed on 14 March 2016]. [Online]. Available: http://westernconsortium.org/

Chase 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.

Thomas J. Rushton graduated with a B.S. in Computer Science and Engineering 2016 from the University of Nevada, Reno. His research interests include virtual reality, data visualization, and artificial intelligence. He is currently working in industry as a software engineer.

Nolan P. Burfield graduate with a B.S. in Computer Science and Engineering in 2015 and is working towards a M.S. in Computer Science and Engineering at the University of Nevada, Reno. He works as a research assistant in the High Performance Computation and Visualization Lab. His research interests are in the areas of computer graphics and financial data computation.


Christine M. Johnson graduated from the University of Nevada, Reno with both a Bachelors of Science and a Masters of Science in Computer Science and Engineering in 2014 and 2015 respectively. She is currently working in industry for Intentional Software Corporation as a software engineer.

Aaron Hesson graduated with a B.S. degree in Computer Science and Engineering at the University of Nevada, Reno in 2014. His specialization is Intelligent Systems.

Daniel Nielson graduated with a B.S. in Computer Science and Engineering from the University of Nevada,Reno in 2014. He also has a degree in English from the University of Southern California. He is currently working in industry as a software engineer.

Bryan Worrell graduated with a B.S.in Computer Science and Engineering at the University of Nevada, Reno in 2014. His interest lies in the field of Games and Simulations and he is currently working as a software engineer in industry.

Donna Delparte is an Assistant Professor in the Department of Geosciences at Idaho State University. She received her BS in Geography (1989) at the University of Regina, Canada and MS (1998) and PhD in Geography (2008) at the University of Calgary, Canada. Dr. Delparte has an extensive background in the applications of GIS and remote sensing to the fields of geosciences, resource management and conservation/environmental planning. She has active research in using Unmanned Aircraft Systems for conservation mapping and precision agriculture applications and analysis.

Tucker Chapman received his B.S. degree in Geology from Brigham Young University, Provo, UT, USA, in 2015. He is currently pursuing a M.S. degree in Geographic Information Science, being advised by Donna Delparte, at Idaho State University, Pocatello, ID, USA and is expecting to graduate in 2017. He is the lead developer of the gridding tool, including performance testing, its REST and Python APIs, and its ArcGIS package.

W. Joel Johansen completed his Masters in GIS in 2015 at Idaho State University and is currently working as a Precision Agriculture Specialist for the J. R. Simplot company. He received a BS in Geology from Brigham Young University in 2013. His current job includes database management, data processing, software development, and analyzing satellite imagery to monitor crop health.

Roger Lew has a B.S. in psychology from University of Idaho in 2004, a M.S. in human factors psychology from University of Idaho in 2007, and a Ph.D. in neuroscience from University of Idaho in 2014. He is currently working as a Research Assistant Professor to manage the Virtual Technology Laboratory and conduct research related to social-ecological systems and nuclear human factors.

Nicholas R. Wood is currently working on his Masters of Science in Integrated Architecture and Design, at the University of Idaho. He received his Bachelors of Science in Virtual Technology and Design at the University of Idaho in 2014. His research interests are integration of data based visualization and Video game narrative.

Matthew Ziegler received his Bachelors of Science in Virtual Technology and Design at the University of Idaho in 2014.

John W. Anderson is an Associate Professor in the College of Art and Architecture, co-founder of the Virtual Technology & Design program and Co-Director of the Virtual Technology Laboratory at the University of Idaho. His teaching and research are focused in the areas of trans-architectures, tele-present environments, evolutionary game theory and design visualization technologies with an emphasis on complex system design and analysis. He is a design thinker who leads interdisciplinary communities of virtual design experts, scientists, engineers, educators, and artists where the focus is the incorporation of virtual technology in all aspects of education, research, modeling, and simulation.

Sergiu M. Dascalu is a Professor in the Department of Computer Science and Engineering at the University of Nevada, Reno, USA, which he joined in 2002. In 1982, he received a Master's degree in Automatic Control and Computers from the Polytechnic University of Bucharest, Romania and in 2001, a Ph.D. in Computer Science from Dalhousie University, Halifax, NS, Canada. His main research interests are in the areas of software engineering and humancomputer interaction. He has published over 140 peerreviewed papers and has been involved in numerous projects funded by industrial companies as well as federal agencies such as NSF, NASA, and ONR.

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.

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

ISCA INTERNATIONAL JOURNAL OF COMPUTERS AND THEIR APPLICATIONS Vol. 23, No. 3, Sept. 2016