Computational Intelligence for Deepwater Reservoir Depositional Environments Interpretation

Tina Yu, Dave Wilkinson, Julian Clark and Morgan Sullivan

Abstract

Predicting oil recovery efficiency of a deepwater reservoir is a challenging task. One approach to characterize a deepwater reservoir and to predict its producibility is by analyzing its depositional information. This research proposes a deposition-based stratigraphic interpretation framework for deepwater reservoir characterization. In this framework, one critical task is the identification and labeling of the stratigraphic components in the reservoir, according to their depositional environments. This interpretation process is labor intensive and can produce different results depending on the stratigrapher who performs the analysis. To relieve stratigrapher's workload and to produce more consistent results, we have developed a novel methodology to automate this process using various computational intelligence techniques. Using a well log data set, we demonstrate that the developed methodology and the designed workflow can produce finite state transducer models that interpret deepwater reservoir depositional environments adequately.

1 Introduction

The petroleum industry is increasingly moving exploration into the deepwater realm to meet the growing demand for oil and gas. Deepwater reservoir projects involve a large amount of financial investment; consequently, business decisions need to be made based on a clear understanding of the producibility of the reservoirs.

One key geologic characteristic that has strong impact on a reservoir's oil recovery efficiency is its depositional environment: the area in which and physical conditions under which sediments are deposited. These include sediment source and supply, depositional processes such as deposition by wind, water or ice, and location and climate, such as desert, swamp or river. Most deepwater reservoirs are deposited in a wide range of depositional systems, and occur at a variety of temporal and spatial scales. Prediction of

net-to-gross, continuity, architecture, and quality of these reservoirs therefore requires integration of seismic, well-log, and core data with appropriate subsurface and outcrop analogs [10].

When conducting such a comparative analysis, it is critical that similar stratigraphic components are compared to one another. The first step in the analysis, therefore, is to identify and label each of the stratigraphic components according to their depositional environments (see Section 2). This interpretation process is labor intensive and can produce different results depending on the stratigrapher who performs the analysis. With the goal of relieving stratigrapher's workload and to produce more consistent results, this research developed a novel methodology to automate this process using various computational intelligence techniques. In particular, we treat the interpretation problem as a language translation problem. The task is to translate a sequence of symbols from one language (reservoir well log data) into another language (the depositional labels).

Finite state transducer (FST) techniques have been widely applied to human language translation, e.g. text-to-speech pronunciation modeling [3] and the parsing of web pages [5]. In this research, we applied an evolutionary computation system to generate FSTs that translate a sequence of reservoir well log data into a sequence of depositional labelsWe also conducted a case study on a deepwater reservoir data set using the developed methods. The results are comparable to that produced by the stratigrapher on this data set. As the project is at the concept-proofing stage, only one data set was provided for us to model the process and to design the workflow. Encouraged by the initial success, we will continue the project and acquire new data sets from other deepwater reservoirs to validate the developed methods.

The paper is organized as follows. Section 2 presents the proposed deposition-based stratigraphic interpretation framework. The developed methodology for stratigraphic interpretation and the designed workflow are then presented in Section 3. Section 4 gives the data set used to conduct our case study. Following the designed workflow step by step, we give detailed explanation of how we carried out the workflow and then present their results in Section 5, 6, 7 and 8.

In Section 5, we describe the segmentation of well log data. In Section 6, we explain the transformation of these segments into fuzzy symbols. In Section 7, we present a co-evolutionary genetic programming system to trains five classifiers. In Section 8, we report the training of FSTs based on the transformed fuzzy symbols and the evolved 5 classifiers. We discuss the limitations of our framework and outline our future work in Section 9. Finally, we conclude the paper in Section 10.

Table 1: Summary of the lithology and GR responses of the 5 different depositional types.

Lithology/Texture	**Gamma ray response**	**Depositional label**
massive sandstone	Sharp based and blocky	Channel-axis (A)
sandstone and shale	Weakly blocky	Channel-off-axis (OA)
mix of sandstone inter-bedded with shale	High	Channel-margin (M)
shale inter-bedded with sandstone	Irregular	Over-bank (OB)
mass wasting (muddy or sandy)	Irregular and chaotic	Mass transport complexes (MTC)

2 A Stratigraphic Interpretation Framework

One systematic approach to identify and label stratigraphic components of deepwater reservoirs is by describing them within a hierarchical framework that is based solely on the physical attributes of the strata and is independent of the thickness and time. In this framework, the fundamental building block of this hierarchical classification is an *element*, defined as the cross-sectional characterization of the volume of sediment deposited within a single cycle of deposition and bounded by an avulsion or abandonment. With this classification scheme, individual elements exhibit a predictable change from axis to margin in grain size, litho-facies type and architectural style. Meanwhile, since avulsion, which is the lateral shifting of a channel or lobe, controls the distribution of these characteristics, elements can be used to understand the distribution of reservoir and non-reservoir facies.

Two or more elements of similar grain size, litho-facies and architectural style form a *complex*. Elements within a complex are genetically related and exhibit a predictable organization and depositional trend. A *complex set* is comprised of either individual complexes of different architectural style and/or complexes of similar architectural style that exhibit depositional trends independent of one another. The description of deepwater sandbodies utilizing this hierarchical approach provides a powerful methodology to directly compare similar stratigraphic components and greatly improve reservoir characterization and the prediction of productivity.

The elements that are of particular interest are *channel*-related as they are the areas where hydrocarbon (oil and gas) deposit. For a finer characterization of a reservoir, we subdivide channel-elements into *channel-axis*,

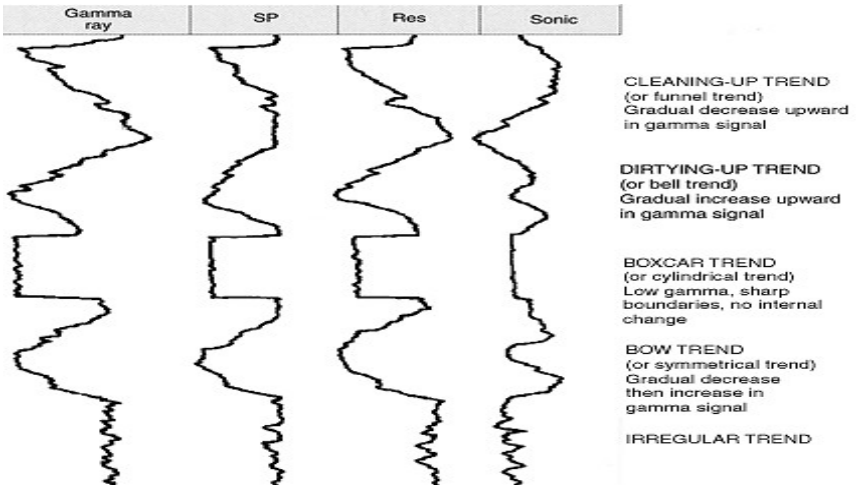


Figure 1: An example of Gamma Ray (GR) well log.

channel off-axis, and *channel-margin* associations.

Channel-axis deposits (A) are dominated by highly-amalgamated, massive sandstones deposited by high-concentration turbidity currents. They normally exhibit a sharp-based, blocky *gamma ray* (GR) response. The channel off-axis association (OA) is composed of stacked, semi- to non-amalgamated, massive to planar-stratified sandstones with inter laminated shales. This type of deposits typically display weakly blocky to moderately serrated GR log character. The channel-margin deposits (M) contain a variety of litho-facies and are characterized by a hetero-lithic mixture of high- and low-concentration turbiditic sandstones interbedded with thick shales. They generally exhibit a serrated and high GR log response.

Two other element types that are non-channel and need to be identified and separated from channel elements are *overbank* and *mass transport complex*. Overbank deposits (OB) are dominated by shale and interbedded with thin sandstones which display an irregular character, lacking a distinct GR log trend. Mass transport complexes (MTC), on the other hand, consist of aggregated components dominated by mass transport. Mass wasting of basin margins and the influx of large quantities of resedimented material may occur at any time as a basin fills. Depending on their source, these complexes can either be very muddy or very sandy, but all tend to be internally chaotic. Due to the lithologic variability of MTC, it is difficult to uniformity characterize their log response, but commonly they display an irregular, chaotic character with an elevated GR response.

Table 1 summarizes the lithology and the GR responses of the 5 deposition types (A, OA, M, OB and MTC). According to this classification scheme, the five deposition types have different responses to GR. It is natural for a stratigrapher to use GR well log data (see Figure 1) as an indicator to classify the 5 types of deposition. The following section presents the workflow we developed to build a computer system that automates this classification task, based on GR well log data.

3 Methodology and Workflow

We applied two computational intelligence techniques, Fuzzy Logic [15] and Evolutionary Computation [4] to train FST models that interpret GR log to identify the 5 different types of deposition. Figure 2 gives the series of steps developed to achieve that goal.

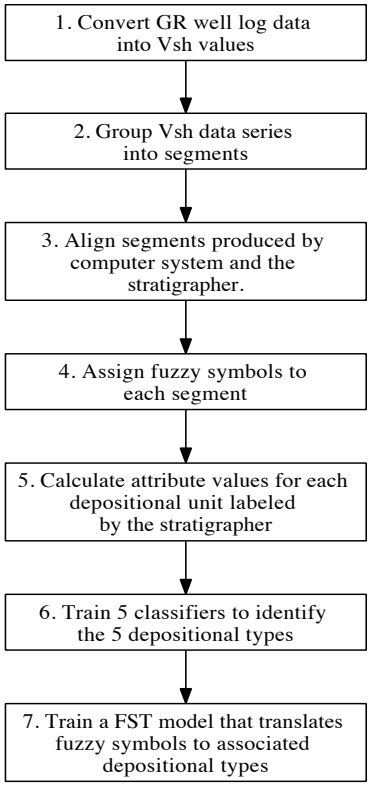


Figure 2: Workflow to construct a computer well-log interpretation system.

In Step 1, GR well log data (whose range is between 0 and 150 API gamma ray units) are converted to *volume of shale* (Vsh) using Equation 1.

$$Vsh_i = \begin{cases} 0, & GR_i < GR_{min}; \\ \frac{GR_i - GR_{min}}{GR_{max} - GR_{min}}, & GR_{min} \leq GR_i \leq GR_{max}; \\ 1, & GR_i > GR_{max}. \end{cases} \quad (1)$$

The purpose of this transformation is to normalize the data while maintaining the trend in the original data. This normalization is necessary for the later step of symbol assignment (Step 4). Note that in Equation 1, GR_{min} is the minimum GR reading in the data set and GR_{max} is the maximum GR reading in the data set.

Step 2 groups the Vsh data series into segments. The segmentation details are provided in Section 5. This step mimics the blocking process (grouping similar consecutive GR data into a segment) that a stratigrapher carries out when performing interpretation. By examining segmented GR data and their associated thickness, a stratigrapher can decide the depositional type of that segment.

Segments produced by computer systems are likely to be slightly different from that blocked by humans as computers can calculate with a high degree of precision while human eyes can not. Normally, the differences are in the form of small edge effects. Since we value computer precision, Step 3 adjusts the boundary locations produced by the stratigrapher to align with that generated by the computer. The adjusted segments will be used in the later steps of classification rules and FST training.

Note that this step is only performed on training data. For new GR well log, this step is skipped and the data preprocessing only consists of Step 1, 2 and 4.

The adjusted segments of Vsh data are represented as a series of numerical values, $\overline{Vsh} = \overline{s_1}, \overline{s_2}, \overline{s_3} \ldots$, where $\overline{s_i}$ is the average of the data values within segment S_i. Numerical values are continuous and less tolerant to uncertainty. By contrast, symbols are discrete and easier for computer to manipulate and to carry out the interpretation task. We therefore simplified the numerical values using symbols. Based on our previous experiences in well log segmentation [12] [13], we decided to use 4 symbols (a, b, c, d) to represent the Vsh data. We first assign each of the 4 symbols with a different Vsh value range. Each $\overline{s_i}$ is then converted to its associated symbol.

To enhance the expressive power of the representation, we used fuzzy, instead of crisp, symbols so that $\overline{s_i}$ values which lay between the boundaries of two symbols can be represented by both symbols (Step 4). The implementation details of fuzzy symbols assignment is provide in Section 6. Using

Table 2: Attributes calculated for each depositional unit.

symbol%	symbol thickness	symbol max
a%	a_ thickness	a_max
ab%	ab_thickness	ab_max
ba%	ba_thickness	ba_max
b%	b_thickness	b_max
bc%	bc_thickness	bc_max
cb%	cb_thickness	cb_max
c%	c_thickness	c_max
cd%	cd_thickness	cd_max
dc%	dc_thickness	dc_max
d%	d_thickness	d_max
variation	total_thickness	no_segments

the fuzzy symbols, the next step is to train FST models that translate these fuzzy symbols to depositional labels.

A FST is a model that maps a string of symbols in a source language into a string of symbols in a target language. The output symbol is determined by two factors: the current input symbol and the current state. However, stratigraphic interpretation is more complicated. In addition to GR reading, a stratigrapher also considers other factors, such as the thickness of each segment and the degree of variation of neighboring segments, to give interpretation. In other words, the output symbol is decided by a model, in addition to the current input symbol and the current state. This model is a decision model that gives one of the 5 possible depositional labels as its output.

We construct the decision model using Genetic Programming (GP) [6]. In order to construct the model that contains similar knowledge as that used by the stratigrapher to classify 5 different deposition types, Step 5 calculates the attribute values listed in Table 2 for every depositional unit labeled by the stratigrapher.

A depositional unit can contain 1 or more segments, hence is represented by 1 or more Vsh symbols. For example, an unit labeled as deposition type `A` can contain 4 symbols `a,ab,b,a`. We used the thickness of each symbol to calculate the attribute values in Table 2. There, the "symbol%" column gives the percentage of each symbol's thickness over the total thickness of the unit. For example, "a%" is the percentage of the thickness of "a" symbols

over the total thickness of the unit. Similarly, the "symbol thickness" column gives the accumulated thickness of each symbol in the unit. The "symbol max" column gives the maximum thickness of each symbol in the unit.

Two attributes that are used to estimate the smoothness of the GR readings in the unit are "no_segments", which is the number of segments in the unit) and "variation", which is the average *distance* of the neighboring symbols in the unit. Here, *distance* is defined as the number of jumps between 2 symbols. For example, the distance between symbols a and dc is 8. Variation of symbol sequence a, ba, dc is (2+6)/2=4. Smoothness of GR readings is an important indication of the deposition type of the unit (see Table 1).

Step 6 uses these attribute information and their associated depositional labels to train 5 classifiers (see Section 7 for implementation details). In Step 7, the 5 classifiers, in addition to the Vsh symbols and their associated thickness, are then used to train a FST as the final model (see Section 8 for the training process details). The final FST model takes a sequence of Vsh symbols representing GR readings and their associated thickness as inputs. It produces a sequence of depositional labels based on the decisions made by the 5 classifiers.

To apply the trained FST model to interpret new GR log data, the GR log data are first transformed into fuzzy symbols following Step 1, 2 and 4. Acting like a stratigrapher, the FST model interprets these symbols and assigns their associated depositional labels.

4 Data Set

One GR well log data set from a deepwater reservoir was provided for us to develop and test our methodology. The GR readings consist of 6,150 data points, each of which was taken at a half foot interval between 4,200 and 7,200 feet under the earth's surface. A stratigrapher has examined the data and assigned 50 depositional labels at different depth level. Following the workflow in Figure 2, we first converted the data into Vsh values. The segmentation of the Vsh data is explained in the following section.

5 Segmentation of Vsh Data

The segmentation process consists of two steps (Step 2 and 3 in the workflow). The first step involves partitioning Vsh data into segments and using

the mean value of the data points within the segment to represent the original data. In order to accurately represent the original data, each segment is allowed to have arbitrary length. In this way, areas where data points have low variation will be represented by a single segment while areas where data points have high variation will have many segments.

There are various approaches to segment data. Due to budget constraint, we only adopted one [7] and modified it to suit our project. This segmentation method starts by having one data point in each segment. That is the number of segments is the same as the number of original data points. Step-by-step, neighboring segments (data points) are gradually combined to reduce the number of segments. This process stops when the number of segments reaches the predetermined number of segments.

At each step, the segments whose merging will lead to the least increase in *error* are combined. The *error* of a segment is defined as:

$$error = \sum_{i=1}^{n} (d_i - \mu)^2 \tag{2}$$

where n is the number of data points in the segment, μ is the mean value of the segment, d_i is the ith data value in the segment.

Although a larger number of segments capture the data trend better, it is also more difficult to interpret. Ideally, we want to use the smallest possible number of segments to capture the trend of the log data. Unfortunately, these two objectives are in conflict: the total *error* of all segments monotonically increases as the number of segments decreases (see Figure 3 (left)). We therefore devised a compromised solution where a penalty is paid for increasing the number of segments. Equation 3 gives the new *error* criterion which is defined as the total error defined previously *plus* the number of segments, N.

$$f = N + \sum_{i=1}^{N} error_i \tag{3}$$

During the segmentation process, the above f function is evaluated at each step. As long as f value is decreasing, the system continues to merge segments. Once f starts increasing, it indicates that farther reducing of the number of segments will sacrifice log characteristics, hence the segmentation process terminates. Figure 3 (right) shows 50 is the optimal number of segments under this compromised approach.

Although simple, the proposed compromised approach to decide the

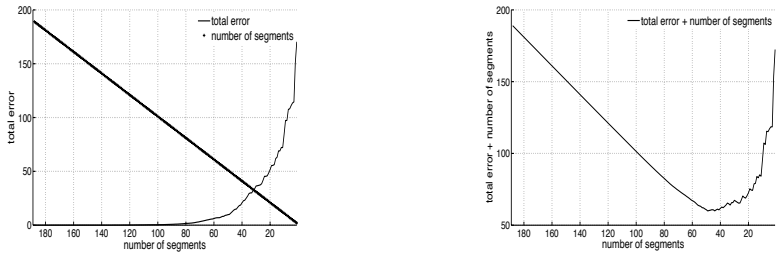


Figure 3: Number of segments vs. total error (left); A compromised solution (right).

number of segments may sacrifice the the capacity to capture the data trend. We will discuss possible improvement and future work in Section 9.

Based on the described scheme, a computer system was developed to carry out the segmentation task. Computer generated segments, however, are not always aligned with the data points where stratigrapher assigns depositional labels. The second step of the segmentation process (Step 3) adjusts the boundary locations of the depositional units to fix the edge effect. In most cases, this requires shifting the labels up or down a few data points.

6 Transform Vsh Data to Fuzzy Symbols

Segmented Vsh data are represented as a set of numerical values, $\overline{Vsh} = \overline{s_1}, \overline{s_2}, \overline{s_3} \ldots$, where $\overline{s_i}$ is the mean value of the data within the ith segment. Step 4 assigns each segment with one of the symbols a, b, c, d according to the following rule:

$$symbol_i = \begin{cases} a, & \overline{s_i} < 0.3; \\ b, & 0.3 \leq \overline{s_i} < 0.5; \\ c, & 0.5 \leq \overline{s_i} \leq 0.7; \\ d, & \overline{s_i} > 0.7. \end{cases} \tag{4}$$

The cut points (0.3, 0.5 and 0.7) are provided by the stratigrapher who interpreted the GR log.

While some segments are clearly within the boundary of a particular symbol region, others may not be so clear. In this case, a crisp symbol does not represent its true value. By contrast, fuzzy symbols use membership function to express the segment can be interpreted as both symbols with some possibilities. Fuzzy symbols are therefore more expressive in this case.

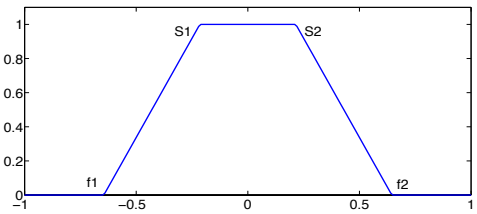


Figure 4: f1, f2, s1, s2, define a trapezoidal-shaped MF.

Table 3: The 4 trapezoidal-shaped MF parameter values.

symbol	f_1	s_1	s_2	f_2
a	0	0.075	0.225	0.375
b	0.25	0.35	0.45	0.55
c	0.45	0.55	0.65	0.75
d	0.625	0.775	0.925	1

In fuzzy logic [15], a membership function (MF) defines how each data point in the input space is mapped into a membership value (or degree of membership) between 0 and 1. The input space consists of all possible input values. In our case, Vsh data have values between 0 and 1. Since we adopted 4 symbols to represent Vsh data, 4 trapezoidal-shaped MFs were designed, one for each of the 4 symbols.

We chose trapezoidal-shaped MF based on our previous experiences [12] and [13], where we found trapezoidal-shaped MF is suitable for the transformation of various types of well log, such as porosity, density and sonic log.

To design a trapezoidal-shaped MF, 4 parameters are required: f_1 and f_2 are used to locate the 'feet' of the trapezoid and s_1 and s_2 are used to locate the 'shoulders' (see Figure 4). We designed these four parameters in the following way. Let t_1 and t_2 be the two cut points that define symbol n and $t_2 > t_1$:

$$y = \frac{t_2 - t_1}{4}$$
$$f_1 = t_1 - y; s_1 = t_1 + y; s_2 = t_2 - y; f_2 = t_2 + y$$

There are two exceptions: symbol a has $f_1 = t_1$ and symbol d has $f_2 = t_2$. Table 3 gives the four parameters for the 4 MFs.

Once the 4 parameters are decided, the MF f for input x is defined in Equation 5. With that, 4 trapezoidal shaped MFs (MF_a, MF_b, MF_c, MF_d)

were defined to transform Vsh data into fuzzy symbols.

$$f(x, f_1, f_2, s_1, s_2) = \begin{cases} 0, & \text{if } x \leq f_1 \\ \frac{x-f_1}{s_1-f_1}, & \text{if } f_1 \leq x \leq s_1 \\ 1, & \text{if } s_1 \leq x \leq s_2 \\ \frac{f_2-x}{f_2-s_2}, & \text{if } s_2 \leq x \leq f_2 \\ 0, & \text{if } f_2 \leq x \end{cases} \tag{5}$$

To fully realize the power of fuzzy logic in enhancing the FST interpretation ability of noisy and uncertain GR data, both the classification rules in Step 6 and the FST in Step 7 have to manipulate fuzzy symbols. Previously, we have implemented fuzzy rules to classify other type of well log data [Yu and Wilkinson, 2007]. However, in this first version of the interpretation system, we started with a crisp version of classification rules and FST model. This is because the fuzzy version of the interpretation system is too complicated for a stratigrapher to comprehend. A simpler crisp version is an ideal first step to introduce the computer interpretation system to stratigraphers and win their acceptance. We will discuss implementing the fuzzy interpretation system in Section 9.

We used the designed 4 MFs to map each segment to one of the following 10 crisp symbols (a, ab, ba, b, bc, cb, c, cd, dc, d) in the following ways. If a segment $\overline{s_i}$ belongs to the region symbol j 100%, i.e. $f(\overline{s_i}, MF_j) = 1$, symbol j is used to represent that segment. If a segment belongs to two symbol regions j and k, we evaluated its degree of membership to the 2 symbol regions. If $f(\overline{s_i}, MF_j) > f(\overline{s_i}, MF_k)$, the segment is mapped to symbol jk. Otherwise, it is mapped to kj.

We implemented the segmentation method described in Section 5 and the symbols transformation algorithm explained in this section in Matlab and applied the software to the Vsh data. The results are presented in the following subsection.

6.1 Results

The 6,150 Vsh data points were segmented and mapped into 62 symbols. They are shown in Figure 5.

We aligned the decision points of the 62 symbols with the positions where the stratigrapher assigned the 50 depositional labels. In the case where one symbol region (segment) has more than one depositional labels assigned to it, we divided the region into multiple segments according to where the

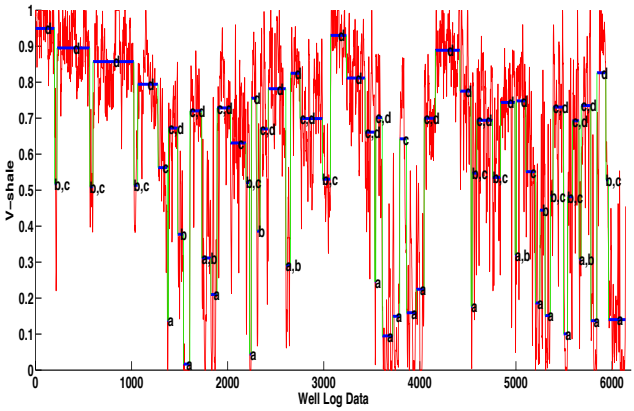


Figure 5: Vsh data transformed into 62 symbols.

position of the label. After this process, the total number of symbol regions (segments) increased from 62 to 82.

The number of symbols (segments) in each depositional unit varies, ranging from 1 to 5. Also, some symbols represent a larger number of data points (thicker) than others. This information is important in determining their deposition types. Step 5 calculates the attribute values listed in Table 2 for each of the 50 depositional units. The resulting data are then used to train 5 classifiers described in the following section.

7 Co-Evolution of Classifiers

Step 6 trains 5 classification rules to identify 5 different deposition types. Among the 50 depositional units, 4 are A, 9 are OA, 14 are M, 19 are OB and 4 are MTC. If we train each classifier individually as a binary classifier, it would be very difficult to obtain good results since the number of depositional units for one class is too small to balance against the number of depositional units for the 4 other classes. An alternative approach is to train the 5 binary classifiers simultaneously. The co-evolutionary computation system described in [13] was developed based on this concept.

In this co-evolutionary system, each binary classifier is represented as a Boolean rule tree. Five populations, each trains one of the five classifiers, are evolved independently and simultaneously. To encourage the co-operation of the 5 populations to evolve the best *team*, the fitness of an evolved individual rule in one population is determined by how well it collaborates with the *best*

rules evolved in the other 4 populations to perform the overall classification task. Here, the *best* rule is the one with the best classification accuracy in the population, based on the fitness function defined in Section 7.1. This co-operative co-evolution model is based on [9] and is illustrated in Figure 6.

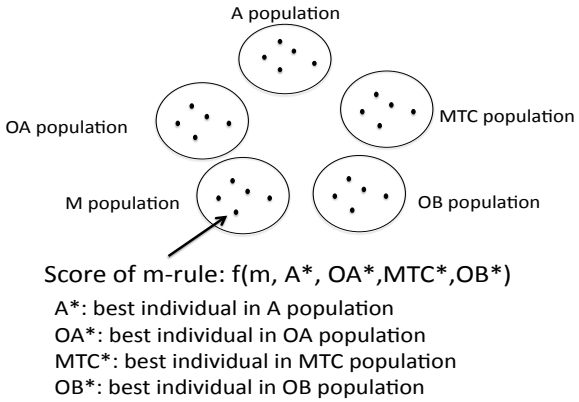


Figure 6: The co-operative co-evolution model.

In terms of implementation, a rule from one population is combined with the best rules from the 4 other populations using an if-then-else template, such as the following:

```
if (OA-rule is evaluated as True)
   then OA
else if (A-rule is evaluated as True)
   then A
else if (MTC-rule is evaluated as True)
   then MTC
else if (OB-rule is evaluated as True)
   then OB
else M.
```

The performance of this combined team determines the fitness of the rule in the population.

At generation 0 when no best rule is known, a rule is randomly selected from each population as the best rule for that population. After that, the best rule is updated at the end of every generation based on the fitness evaluation, so that a good rule can be immediately used to combine with rules in other populations in the following generation and impact evolution.

With 5 rules in each team, there are 5! ways to combine (order) them using the if-then-else template. Since rules applied earlier in the if-then-else template are weighted more than rules that are applied later in the classification decision, it is important to acquire a suitable rule sequence for a team to achieve good classification accuracy.

To learn the best rule order strategy, we used a simple hill-climbing approach. At the beginning of each experimental run, a random order of rule sequence is generated (e.g. OA, A, MTC, OB, M). This order is used to combine rules for fitness evaluation at generation 0. At the end of generation 0, the best rule from each of the 5 populations are selected. Meanwhile, the current order sequence is mutated to obtain a new rule order sequence. If the team of the 5 best rules combined using the new order sequence gives better or equal classification accuracy than that produced by the team combined based on the old sequence, the new sequence is adopted and used in the next generation for rule combination. Otherwise, the old sequence is retained and reused in the next generation. This process is repeated at the end of each generation to determine the rule order strategy in the next generation.

7.1 Experimental Setup

We implemented the co-evolutionary system using a GP software called PolyGP [11]. GP [6] applies genetic algorithms [4] to evolve program trees. In our case, the program trees are Boolean classification rules. When a Boolean rule is executed, it returns a value of $true$ or $false$. If the value is $true$, the classification is positive. Otherwise, the classification is negative.

We provided six operators for GP to construct the classification rules: *and*, *or*, *nor*, *nand*, $<$ and $>$. They can combine any subset of the 33 attributes listed in Table 2 and the following constants to form a Boolean rule:

- Boolean constants: `true` and `false`;
- integer constants: 1 – 10;
- percentage constants: 0.0 – 1.0;
- double constants: 0.0 – 250.0;

Figure 12 gives an example of a Boolean rule that classifies deposition type A.

The attributes and the constants have different types, while each operator can only be applied to a certain type (see Table 4).

Table 4: Operators, attributes, constants and their types.

name	type
and	$bool \rightarrow bool \rightarrow bool$
or	$bool \rightarrow bool \rightarrow bool$
nand	$bool \rightarrow bool \rightarrow bool$
nor	$bool \rightarrow bool \rightarrow bool$
$<$	$a \rightarrow a \rightarrow bool$
$>$	$a \rightarrow a \rightarrow bool$
symbol%	*percentage*
symbol_thickness	*double*
symbol_max	*double*
variation	*integer*
no_segments	*integer*
% constants	*percentage*
integer constants	*integer*
double constants	*double*
boolean constants	*bool*

To assure only semantically meaningful rules are generated, the PolyGP employed a type system to performs type checking. For example, $<$ can be applied to compare attributes with the same type, e.g. `d%` with `b%` , `a_thickness` with `d_thickness` and so on. The a in Table 4 is a type variable which can be substituted with any concrete type, such as `percentage` and `integer`.

To work with the Boolean rule tree representation, we employed four genetic operators in this study: homologous crossover, *and*-crossover, *or*-crossover and mutation. Homologous crossover selects common location in both parent trees to carry out the crossover operation and produce two offspring (see Figure 7).

The *and*-crossover combines two parent rules into one rule using the *and* operator (see Figure 8(A)). The *or*-crossover combines two parent rules into one rule using the *or* operator (see Figure 8(B)). The mutation operation can perform sub-tree, function and terminal mutations, depending on the selected mutation location (see Figure 9). Mutation produces single offspring.

Table 5 lists the GP parameter values used to conduct the experimental

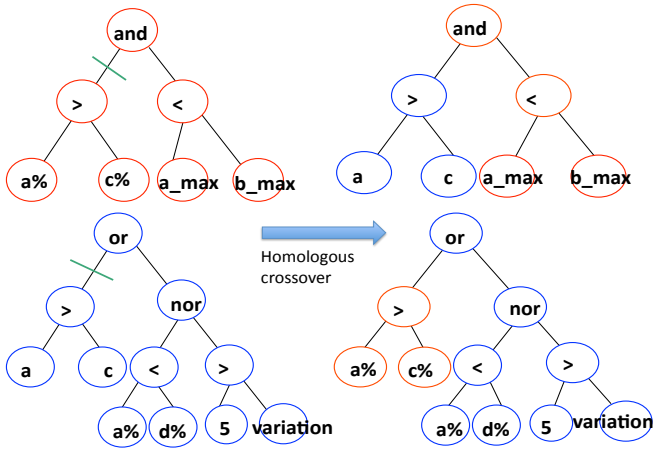
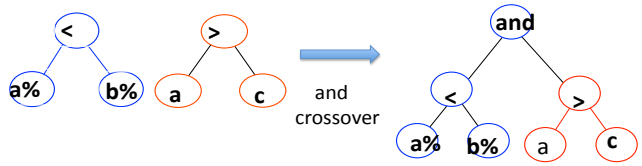
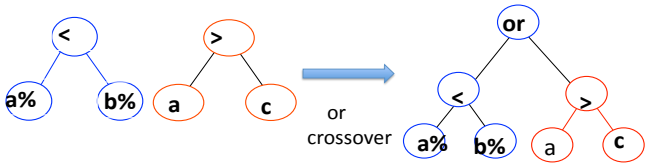


Figure 7: The homologous crossover operator.



(A) The AND-crossover



(B) The OR-crossover

Figure 8: (A)*and*-crossover (B) *or*-crossover.

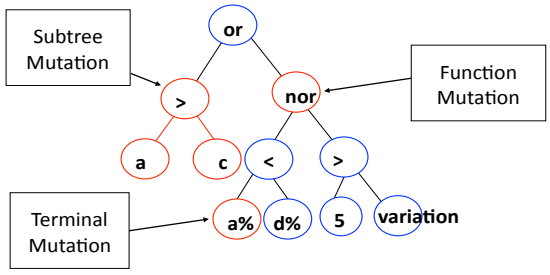


Figure 9: The mutation operator.

runs. A population consists of 100 Boolean rules. At the beginning of each

Table 5: GP parameters for experimental runs.

parameter	value	parameter	value
tournament selection	size 2	homoXover rate	30%
max tree depth	6	orXover rate	30%
population size	100	mutation rate	30%
max generation	200	andXover rate	30%
number of runs	50	elitism	1

new generation, the rule with the best fitness in the current population is copied over to the new population (elitism = 1). The rest 99 Boolean rules are then generated by selection and genetic operations as described below.

A rule is first selected (using tournament selection of size 2) from the current population. Next, one of the 4 genetic operations is applied based on the following probabilities:

```
if (random() < 30%)
   perform homologous crossover
else if (random() < 30%)
   perform mutation
else if (random() < 30%)
   perform orCrossover
else if (random() < 30%)
   perform andCrossover
else
   perform copy
```

To carry out any of the 3 crossover operations, another rule is selected using the same tournament method.

The fitness of a rule is the classification accuracy of the combined if-then-else rule team that the rule is a part of. When the team gives a correct interpretation for a depositional unit, it is a hit. The number of hits divided by the number of depositional units in the training data (50) is the fitness of the evaluated rule.

Additionally, we used penalty to discourage generating large size rules. Initially, all rules at generation 0 have the same depth of 6, hence $2^6 - 1 = 63$ nodes. After that, the rule trees tend to grow in size. A rule whose tree size (l) is larger than 150 is penalized. The final fitness of a rule is therefore:

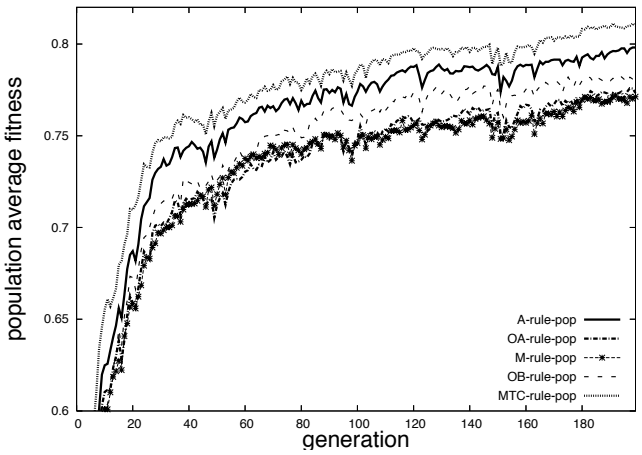


Figure 10: Average fitness of the 5 co-evolved populations.

$$fitness = \frac{\sum_{i=1}^{N} hit_i}{N} - (\frac{l-150}{l})^2, \quad \text{N=50}$$

Among the 5 depositional types, A and OA are close to each other while M, OB and MTC are similar in their geological characteristics. To promote rules that produce less serious misclassification, the distance between the target label and the classified label is used as the secondary criterion for selection. Here, the 5 depositional labels are ordered as follows: A, OA, M, OB, MTC. The *distance* between two depositional labels is the number of jumps between them. For example, the distance between A and OA is 1 and the distance between A and MTC is 4. If two rules have the same fitness value, the one that gives a smaller distance error is the winner during tournament selection.

7.2 Results

We performed 50 experimental runs. Figure 10 gives the average fitness of the 5 co-evolved populations. It shows that all 5 populations improved consistently throughout the runs. Among them, MTC-rule population has the highest average fitness. This is followed by the A-rule and OB-rule populations. The M-rule and OA-rule populations have the lowest average population fitness.

These results suggest that MTC-rule has the least impact on the overall classification results. Once *best* rules from other populations are integrated into a team, any MTC-rule in the population can produce good classification

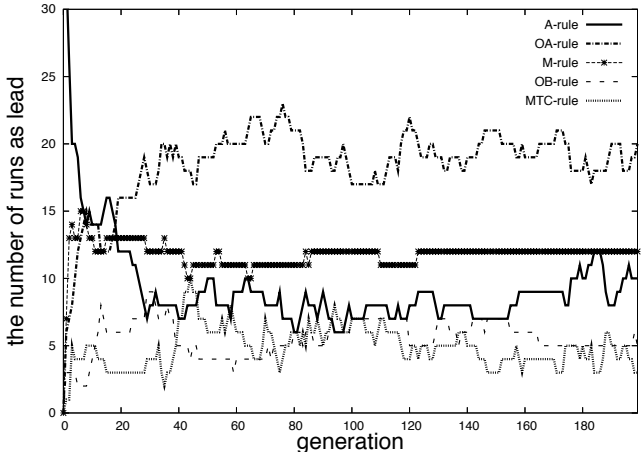


Figure 11: The number of runs that each rule is selected as the lead.

results. We examined the number of runs that each rule was selected as the lead classifier in the team (see Figure 11) and found that MTC-rule is the least selected one, indicating that it is of the least impact. This result is consistent with our hypothesis.

The two most selected rules as the lead classifier in the team are OA-rule and M-rule. Figure 10 also shows that these two populations have the lowest average fitness. This indicates that the *best* rules from other populations are not sufficient for the team to produce good classification accuracy; it is critical for a team to have good OA and M rules to produce good classification results. In other words, OA-rule and M-rule have more impact on the overall classification results than other rules.

One factor that might have contributed to this evolutionary dynamics is the number of training data in each class. In general, training process biases toward the class that has a larger number of training data. As a result, that classification rule has a higher impact on the overall classification results. In this data set, MTC class has the smallest number (4) of data points while OA and M classes have the largest number (14 and 19). It is understandable that the training process ignored MTC-rule but focusing on evolving good OA and M rules for the team to produce good overall classification accuracy.

The best team produced from the 50 runs gives classification accuracy of 90%: 5 of the 50 depositional units were mis-classified. Table 6 gives the details.

Among the 5 mis-classified depositional labels, two MTC were mis-

Table 6: Classification accuracy of the best team.

	A	OA	M	OB	MTC	A	OA	M	OB	MTC
A	4	0	0	0	0	100%	0%	0%	0%	0%
OA	0	9	0	0	0	0%	100%	0%	0%	0%
M	0	0	13	1	0	0%	0%	93%	7%	0%
OB	0	0	1	18	0	0%	0%	5%	95%	0%
MTC	0	0	1	2	1	0%	0%	25%	50%	25%

labeled as OB, one MTC was mis-labeled as M, one OB was mis-labeled as M and one M was mis-labeled as OB. As mentioned previously, M, OB and MTC have similar depositional ingredients. These mis-classifications, hence, are not considered to be serious.

Figure 12 gives the A-rule tree in the best team. Its interpretation is straight-forward: symbol a (which represents the smallest Vsh value) has to dominate the segments in the depositional unit. This model accurately describes the composition of channel-axis deposition: massive sandstone without shale (see Section 2). The other 4 rules in the best team are given in Figures 13, 14, 15 and 16. Note that the symbols in the classification rules are shorthand for the symbol thickness. For example, `a` is the shorthand for `a_thickness`; `cd` is the shorthand for `cd_thickness`.

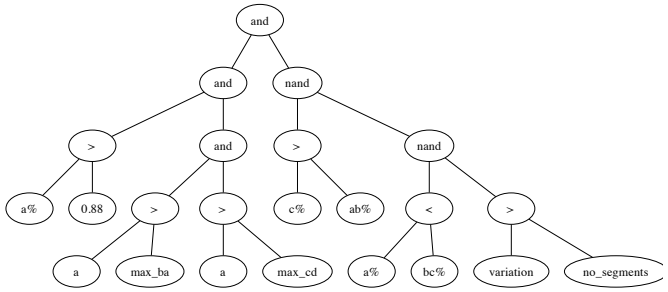


Figure 12: The "A" classification rule.

Not all 33 attributes in Table 2 are used in these rules to classify deposition types. For example, the thickness of symbol `b` does not appear in any of the rules. However, this does not mean the classification decision is independent of the presence of symbol "b" in the Vsh data. This is because the presence/absence of symbol `b` has impact on the attributes database values. For example, the increases of thickness of symbol "b" decreases the percent-

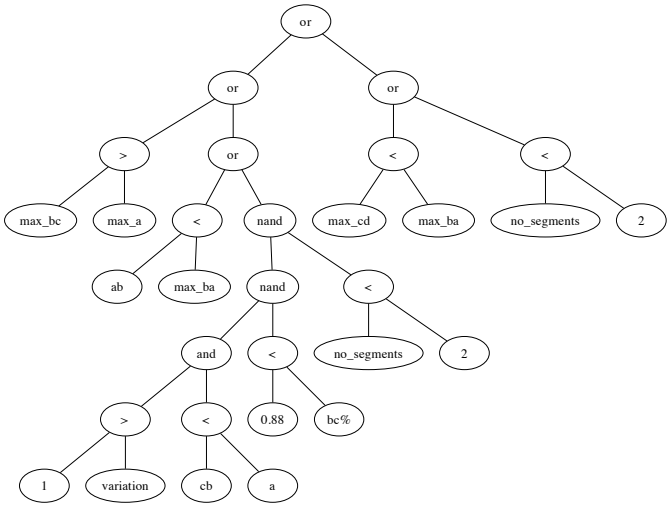


Figure 13: The “OA“ classification rule.

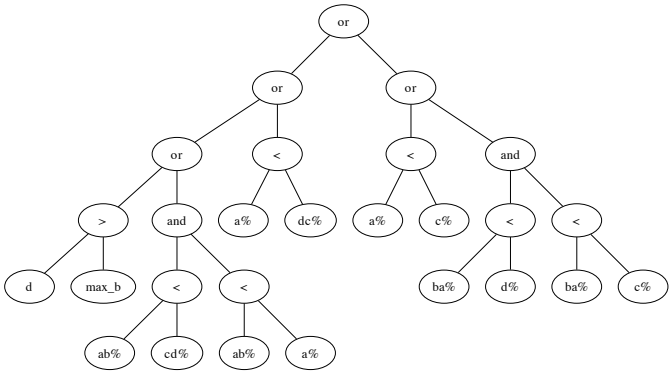


Figure 14: The “M“ classification rule.

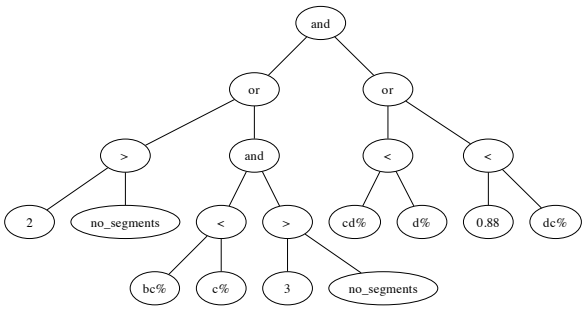


Figure 15: The "OB" classification rule.

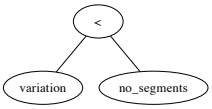


Figure 16: The "MTC" classification rule.

ages of other symbol thickness (`a%`, `c%`). Meanwhile, the `total_thickness` is increased and the number of segments (`no_segments`) is increased. Since these values are used by the the classification rule to make classification decision, symbol `b` has influence on the classification results.

Similarly, symbol "b" influences the FST interpretation in the following ways.

- The symbol is used by the FST to select one of the 5 rules to fire;
- The symbol is used to decide the next state of FST, which has impact on the next classification rule to fire after reading the next Vsh symbol.

Note that when applying these 5 classifiers to a new GR well log, we have to make sure that the well log is from a reservoir which has similar lithology as that of the reservoir from which the GR training data was obtained. Otherwise, these classifiers would not work well.

8 Evolving Finite State Transducers

With the establishment of the 5 classifiers, we can proceed to the last step (Step 7) of FSTs training. This FST translates a sequence of Vsh symbols to a sequence of depositional labels.

There are various ways to represent a FST for evolutionary learning, such as the pioneer work by [2] and extended work by [1]. In this research, we followed the work of [8] and used two tables to represent a FST. The first table is transition table (see Figure 8), which gives the next state of the FST based on the current input symbol and the current state. The second table is output table (see Figure 9), which gives the output symbol of the FST also based on the current input symbol and the current state. In our case, the input symbols are the 10 Vsh symbols and the output symbols are the names of the 5 classification rules.

After a classification rule is decided by the FST, it is applied to the current attribute database. If it returns `true`, the name of the rule is the output depositional label. Otherwise, the output label is `null`. Consequently, the length of the output deposition labels (e.g. `A, OA, M, OB, MTC or null`) generated by the FST is always the same as the length of the input Vsh symbols. The following sub-section describes the workflow of this FST.

8.1 FST Operating Procedures

As shown in Figure 17, the FST is initially at state 0 (`S0`). At each step, one Vsh symbol and the thickness of the segment the symbol represents are processed. The thickness is used to update the attribute values in the database listed in Table 2. For example, the number of segments (`no_segments`) is increased by 1 and the `total_thickness` is increased by the symbol's thickness. Meanwhile, based on the current state and the Vsh symbol, a classification rule in the output table is selected to fire. The name of the selected rule is the proposed deposition type for the sequence of segments that have been processed so far and assigned with `null` labels.

After the selected rule is applied to the updated attribute database, it returns a value of `true` or `false`. If the value is `true`, it indicates that the sequence of segments and their associated thickness satisfy the criteria of the proposed deposition type. The depositional label associated with the classification rule is assigned to the current Vsh symbol. Once this happens, all attributes values in the database are set to 0. The system is ready to process the next Vsh symbol and decides what the next depositional label is.

If the classification rule returns `false`, it indicates that the sequence of segments and their associated thickness do not satisfy the criteria of the proposed deposition type. The `null` label is assigned to the current Vsh symbol. The current attribute database is retained as it reflects the thickness information of all processed Vsh symbols that have been assigned with `null`

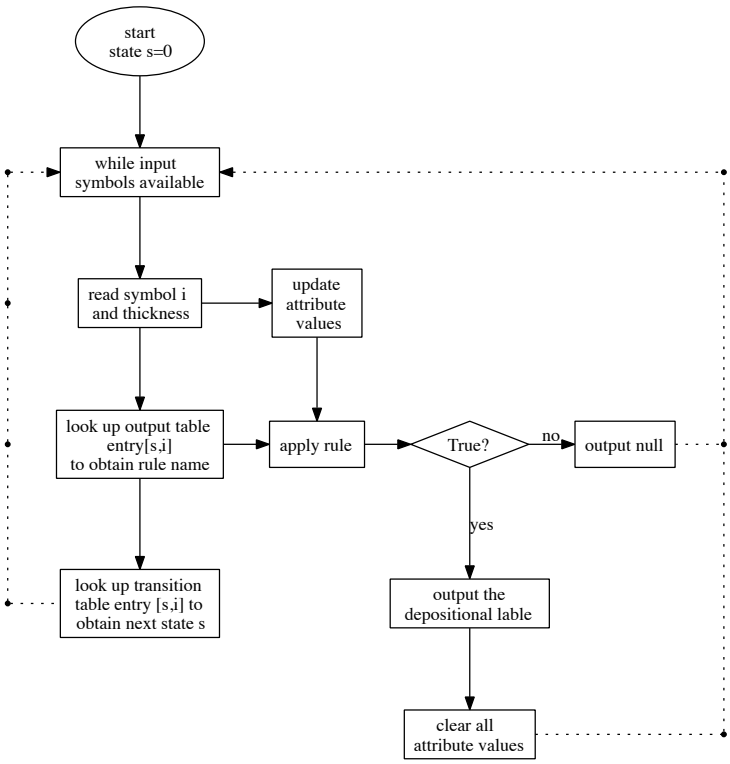


Figure 17: The operating procedure of the FST.

labels. The FST is ready to process the next Vsh symbol and utilizes the accumulated information in the attribute database to decide what the next depositional label is.

Regardless of the rule firing result, the FST moves to a new state. The name of the next state is given by the transition table based on the Vsh symbol and the current state. After moving to the new state, the FST continues processing the next Vsh symbol and repeats the operation until all Vsh symbols are exhausted.

Using the transition table in Table 8 and the output table in Table 9, we show the FST operation on input sequence: $(d, 96), (d, 3), (cb, 12.5)$.

With the initial state of `S0` and the Vsh symbol d, the proposed classification rule is OB, according to the output table. After updating the attribute database with the thickness of 96, OB-rule is applied to the database. Assume the rule returns `true`, OB becomes the depositional label assigned to symbol d. After that, all attribute values in the database are cleared (become 0). The FST then moves to next state (`S18`) according to the transition table.

Next, symbol d is processed and the attribute database is updated with the thickness information (3). The proposed rule, according to the output

table, is OA. Assume the rule returns `false` on the updated database, *null* is assigned to symbol d. The FST moves to the next state (`S10`) without clearing the attribute database.

The next symbol to process is *cb*. After the attributes database is updated with the thickness of 12.5, it reflects the characteristics of two segments (d and *cb*). The proposed classification rule, according to the output table, is A. Assume the rule returns `true` on the current attribute database, depositional label A is assigned to symbol *cb*. The FST clears the attributes database and moves to next state (`S7`). Since there is no more Vsh symbol, the interpretation process terminates. The depositional labels sequence produced by the FST on the given Vsh symbols is $(OB, null, A)$.

8.2 The Evolutionary System

The two tables in the FST have the same size of $N_Q \times N_I$, where N_Q is the number of states and N_I is the number of input symbols. In this work, N_I is 10: `a...d` and N_O is 5: `A, OA, M, OB, MTC`. Based on our estimation of the task complexity, we chose $N_Q = 20$, which seems to be reasonable or the FST to carry out the inputs/outputs translation. This decision is ad hoc and requires more research to justify its appropriateness (see Section 9 more more discussion).

Based on these parameter values, the number of possible FSTs is:

$$S = N_Q^{N_Q \times N_I} \times N_O^{N_Q \times N_I} \approx 10^{400} \tag{6}$$

This is a huge search space and stochastic search methods such as evolutionary algorithms are good candidates to locate a decent solution within a reasonable time frame. We therefore implemented a GA in Java to train the two FST tables [14].

Unlike the traditional GA [4], which evolves vector representation, this GA evolves 2-dimensional tables. We therefore have to modify the genetic operators to work with the table representation.

This GA only uses mutation operator, which works the same way as that in [8]. First, a decision is made with equal probability to either mutate the transition table or the output table. A random location is then selected in the chosen table, and the entry there is modified. This ensures that mutation causes at least one change in one of the two tables.

After that, an iteration is performed over all the table entries apart from the entry just modified, changing each entry with a probability of $\frac{1}{N_Q \times N_I}$. When modifying an entry, a symbol (state name or rule name, depending

on the modified table) is chosen uniformly from all possible symbols except the current one. According to [8], a single call to the mutation operator is most likely to produce one or two changes to the two FST tables, but can also produce more. The probability that this mutation is applied to the population is 2%.

The fitness of a FST is calculated based on the depositional labels it produces. After processing a Vsh symbol, a FST produces a depositional label or *null*. The length of the produced labels, hence, is always the same as the length of the Vsh symbols.

We first aligned the 82 Vsh symbols with the 50 depositional labels produced by the stratigrapher, based on the positions where the depositional labels were given. We then inserted `null` among the 50 labels on Vsh symbol positions where the stratigrapher did not give a label. With that, the length of the original 50 depositional labels produced by the stratigrapher is increased to 82.

Next, we aligned this label sequence with that produced by the FST and the number of mis-match between the two is the FST's fitness. A FST that produce all 50 deposition labels correctly at the correct segment position has fitness value 0.

$$fitness_{FST} = \sum_{i=1}^{N} mismatch_i, \quad \text{N=82.}$$

To encourage FSTs to produce less serious mis-match, we used the same distance measure used to co-evolve classification team as the secondary criterion for selection (see Section 7.1). If two FSTs have the same number of mis-match, the one with a smaller distance error is the winner during selection. Table 7 lists the parameter values used to conduct 100 experimental runs.

Table 7: GA parameters for experimental runs.

parameter	value	parameter	value
tournament selection	size 2	elitism	size 1
population size	50	N_Q	20
max generation	1000	mutation rate	2%

8.3 Results

Figure 18 gives the average and the best fitness of the population averaged over 100 runs. It shows that the populations improved consistently throughout the run of 1,000 generations. The best FST gives 7 mis-matches to the 82 depositional label produced by the stratigrapher:

- two M were mis-labeled as OA;
- one A was mis-labeled as OA;
- two OB were labeled as *null*. The segments were then combined with the consecutive segments and labeled as M, which was the correct label;
- two OB were mis-labeled as M.

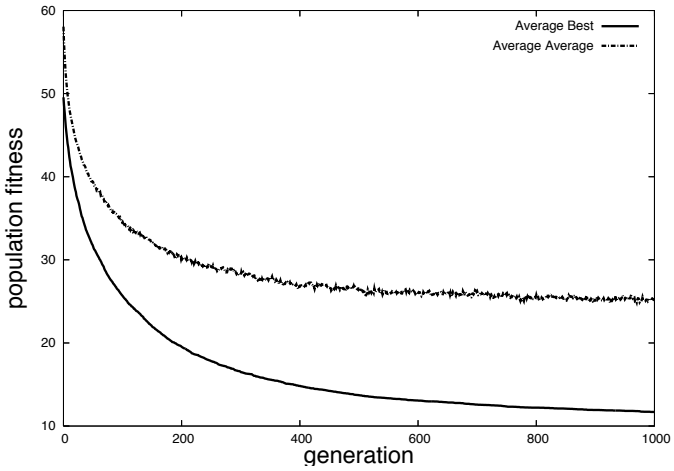


Figure 18: Average and best population fitness.

As mentioned in Section 7.1, A and OA have similar lithology while M and OB are close with their depositional ingredients. The FST translation results are therefore comparable to that produced by the stratigrapher on this data set. We list this best transition table and output table in Table 8 and Table 9.

Table 8: The best finite state transducer transition table.

input	a	ab	ba	b	bc	cb	c	cd	dc	d
S0	S8	S2	S19	S9	S1	S14	S11	S7	S18	S18
S1	S9	S17	S4	S5	S3	S2	S14	S12	S2	S10
S2	S9	S18	S1	S10	S3	S9	S16	S4	S1	S3
S3	S15	S9	S15	S0	S16	S13	S14	S17	S16	S2
S4	S0	S0	S17	S8	S7	S9	S3	S6	S6	S13
S5	S14	S12	S9	S0	S14	S16	S6	S3	S3	S8
S6	S1	S14	S12	S19	S3	S1	S16	S1	S3	S13
S7	S17	S19	S4	S19	S3	S10	S6	S5	S15	S15
S8	S12	S6	S5	S13	S16	S1	S4	S14	S16	S3
S9	S3	S19	S4	S19	S11	S1	S2	S15	S16	S8
S10	S7	S9	S19	S6	S16	S7	S11	S15	S7	S6
S11	S4	S13	S19	S18	S10	S8	S19	S15	S2	S12
S12	S19	S1	S6	S14	S11	S9	S3	S18	S3	S10
S13	S10	S11	S10	S11	S7	S8	S3	S15	S17	S6
S14	S9	S16	S0	S3	S4	S3	S8	S15	S5	S3
S15	S13	S13	S3	S6	S9	S8	S3	S7	S18	S6
S16	S18	S6	S2	S5	S0	S14	S10	S14	S11	S4
S17	S9	S16	S4	S6	S7	S6	S13	S7	S9	S4
S18	S1	S12	S19	S6	S2	S9	S0	S0	S5	S10
S19	S13	S2	S15	S18	S14	S0	S18	S2	S12	S0

9 Limitations and Future Works

Using one GR well log data set, we demonstrated that the developed methodology and the designed workflow can produce FST models that interpret deepwater reservoir depositional environments adequately. However, the applicability of the method to the deepwater reservoir industry is still unknown until we have tried and test it on several more data sets. Additionally, we need to investigate some practical issues to assure the framework works for a wide range of deepwater reservoirs.

First, the data set we used to develop the method is reasonably clean, which helped train good quality models. It is not clear if the method would work well on noisy data. The proposed interpretation system is a fuzzy model, which should perform well on noisy and uncertain data. However, we have only implemented the crisp version of the system. To fully realize the fuzzy version of interpreter, the 5 classifiers and the FST need to manipulate fuzzy symbols. For example, we need to provide fuzzy definition for the 33 attributes and the 6 Boolean operators. Similarly, the FTS should decide the next state and the classification rule to fire based on the fuzzy Vsh symbol. We plan to develop the fuzzy version of the interpretation system in our future work.

Table 9: The best finite state transducer output table.

input	a	ab	ba	b	bc	cb	c	cd	dc	d
S0	OA	OB	OB	OB	OA	A	MTC	M	OB	OB
S1	OA	MTC	OB	MTC	MTC	M	M	M	OB	M
S2	OB	OA	MTC	OA	MTC	M	OB	M	OA	A
S3	M	OA	OA	A	M	MTC	OB	M	OB	OB
S4	M	MTC	A	OB	M	OB	OA	OB	MTC	OB
S5	M	A	M	OA	M	OA	A	A	MTC	A
S6	A	M	OA	MTC	MTC	OA	OB	OA	M	A
S7	OA	M	M	OB	M	A	M	OA	MTC	M
S8	OA	OB	MTC	MTC	OA	OB	MTC	MTC	M	M
S9	OA	OA	A	A	MTC	MTC	MTC	M	OA	MTC
S10	OA	A	OA	OA	OB	A	MTC	OA	A	OA
S11	MTC	OB	OB	OB	A	MTC	M	A	MTC	OA
S12	M	M	M	MTC	OA	A	OB	MTC	MTC	OB
S13	OA	MTC	M	M	OA	MTC	OB	OB	OA	OA
S14	A	MTC	MTC	MTC	MTC	OA	OB	M	MTC	M
S15	M	OB	OB	A	MTC	A	OB	MTC	A	OB
S16	OA	MTC	A	MTC	MTC	OB	M	MTC	OA	OA
S17	A	OA	A	OA	M	OA	OB	OB	M	M
S18	A	MTC	OB	OA	MTC	A	OB	MTC	OB	OA
S19	A	MTC	A	A	OA	OB	MTC	A	OA	MTC

Second, with 6,150 data points, the computation time to develop the interpretation system manageable. It took a single-core Linux machine seconds to complete data segmentation and symbol mapping (Step 1–4). The machine took 2 hours to train 5 classifiers (Step 6) and 2 hours to train the FST (Step 7). We are not clear if the method would work with more or less amount of data. Our hypothesis is that if we apply the method to multiple GR logs from more than one reservoirs at the same time, the resulting models should be better and more robust. We will test this hypothesis in our future works.

There are many areas in the proposed framework that we can improve. First, the proposed compromised approach to decide the number of segments may not be optimal. Although the number of segments could not be too large due to the computational complexity, the capacity of capturing the data trend is also important. In our future work, we will assign weight factors to the two objectives of `error` and `the number of segments` in Equation (3). We will evaluate the trade-off by adjusting the weights to segment the well logs to identify a good balance.

Second, the FST is a one-pass algorithm without back track. After reading a Vsh symbol, the FST gives the depositional label based on the selected classification rule and the current attribute database.This interpretation decision can not be revised later. It might be interesting to have a back track FST that can revise previous interpretation if it finds a better interpretation.

We will research this alternative FSM in our future work.

Next, the FST performs one look-ahead of the new Vsh symbol to make interpretation decision. One-look-ahead is the simplest FST. It might be interesting to extend the FST to look ahead more than one Vsh symbol at a time to make interpretation decision. We will investigate this alternative FSM in our future work.

Last, the FST tables show that certain sets of inputs and certain sets of states have similar behavior. This suggests that there might be room for aggregation by reducing the input size (N_I) and state size (N_Q) of the FST. In our future work, we will investigate the granularity of the FST.

10 Concluding Remarks

Reservoir characterization continues to present new challenges as exploration moves to new territories, such as deepwater. We have developed a stratigraphic interpretation framework that analyzes depositional information to improve reservoir characterization. In this framework, one critical step is the interpretation of the stratigraphic components in the reservoir. The quality of the interpretation can impact the prediction of oil recovery of the reservoir.

This research developed a novel method and designed a workflow to automate this interpretation process. In particular, we treated the interpretation task as a language translation problem and applied various computational intelligence techniques to train FST models that carries out the interpretation task. To the best of our knowledge, this is the first time the language-translation approach with a FST model is used to interpret well log data. We also showed the interpretation resulst produced by the FST are comparable to that produced by the stratigrapher on this data set.

With this encouraging initial result, we will continue our future work on the areas outlined in Section 9. We will also help stratigraphers to understand the FST model training process to gain their acceptance of the developed models.

References

[1] D. Fogel, *Evolutionary Computation: Toward a New Philosophy of Machine Intelligence*, IEEE, 1995.

[2] L. J. Fogel, A. J. Owens and M. J. Walsh, *Artificial intelligence through Simulated Evolution*, Wiley, 1966.

[3] D. Gildea and D. Jurafskyd, Automatic Induction of Finite State Transducers for Simple Phonological Rules. In *Proceedings of the 33rd Conference on Association for Computational Linguistics*, pp 9–15, 1995.

[4] J. Holland, *Adaptation in Natural and Artificial Systems*, MIT Press, 1975.

[5] C.-N. Hsu and M.-T. Dung, Generating Finite-State Transducers for Semi-Structured Data Extraction from the Web. *Information Systems*, vol. 23, no. 8, pp. 521-538, 1998.

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

[7] J. Lin, E. Keogh, S. Lonardi and B. Chiu, A Symbolic Representation of Time Series, with Implications for Streaming Algorithms. In *Proceedings of the 8th ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery*, 2003.

[8] S. M. Lucas and T. J. Reynolds, Learning Finite State Transducers: Evolution Versus Heuristic State Merging. *IEEE Transactions on Evolutionary Computation*, vol. 11 no. 3, pp. 308-325, June 2007.

[9] M. A. Potter and K. E. De Jong, A Cooperative Coevolutionary Approach to Function Optimization. In *Parallel Problem Solving from Nature – PPSN III*, pages 249–257, Berlin, 1994. Springer.

[10] G. Shanmugam, Deep-water Processes and Facies Models : Implications for Sandstone Petroleum Reservoirs, Elsevier:Oxford, 2006

[11] T. Yu and C. Clack, PolyGP: A Polymorphic Genetic Programming System in Haskell. In *Genetic Programming 1998: Proceedings of the Third Annual Conference*, pages 416–421. Morgan Kaufmann, 1998.

[12] T. Yu and D. Wilkinson, A Fuzzy Symbolic Representation for Intelligent Reservoir Well Logs Interpretation. In O. Castillo, P. Melin, J. Kacprzyk and W. Pedrycz, editors, *Soft Computing for Hybrid Systems*, pages 417–426, Springer Verlag, 2008.

[13] T. Yu and D. Wilkinson, A Co-evolutionary Fuzzy System for Reservoir Well Logs Interpretation. In T. Yu, L. Davis, C. Baydar, and R. Roy,

editors, *Evolutionary Computation in Practice*, Chapter 9, pages 199–218, Springer, 2008.

[14] T. Yu, D. Wilkinson, J. Clark and M. Sullivan, Evolving Finite State Transducers to Interpret Deepwater Reservoir Depositional Environments. In *Proceedings of Congress on Evolutionary Computation*, pages 3490 – 3497, IEEE, 2008..

[15] L. A. Zadeh, Fuzzy sets. *Information and Control*, 8: 338 – 353. 1965.