



.....

**INSTITUTO POLITÉCNICO NACIONAL**

**CENTRO DE INVESTIGACIÓN EN COMPUTACIÓN**

**ANÁLISIS DE LA SIMILITUD ENTRE  
PROGRAMAS DE ALTO NIVEL**

**TESIS**

**QUE PARA OBTENER EL GRADO DE  
MAESTRO EN CIENCIAS DE LA COMPUTACIÓN**

**PRESENTA**

**MIGUEL ANGEL MIRÓN BERNAL**

**DIRECTORES DE TESIS**

**DR. HUGO C. COYOTE ESTRADA**

**DR. JESÚS G. FIGUEROA NAZUNO**



**MÉXICO, D.F.**

**FEBRERO 2008**

## **Dedico este trabajo**

*A Ti, por permitirme conocerte y amarme en la forma que lo haces.*

*Sal 143,8*

*A mi madre, **Asunción Bernal Pérez**, por su tenacidad,  
confianza y cariño, que a través de los años forjaron  
la persona que hoy soy. Gracias Mamá.*

*Rom 8,37-39*

*A mi padre, **Jorge Mirón Tenorio**, por  
darme libertad en pensamiento, por sus consejos y apoyo.*

*Gracias Papá.*

*Pro 23,22 -25*

*A mi hermano, **Ricardo Admin. Mirón Bernal**, por ser parte de mi  
felicidad y tesoro invaluable, sigue siempre adelante. Te quiero.*

*1Tim 4,12-16*

*A **Asunción Pérez Melo** y **Rosaura Bernal Pérez**,  
por su cariño y por sus consejos, gracias por tus oraciones abue.  
*Col 1,3-5, Sir 1,23-24**

*A **Georgina Bernal Pérez**, por su confianza y motivación.*

*Sir 35,16-18*

*A **Fabiola Colorado Rodríguez**, por su apoyo y cariño incondicional,  
sabes muy bien lo que significas para mi, gracias por tu ternura.*

*Ec 4,9-12*

## **AGRADECIMIENTOS**

*Al Dr. Hugo Coyote Estrada, por la confianza otorgada, y la libertad para emprender las ideas que llevaron a este trabajo.*

*1Co 3,8-9*

*Al Dr. Jesús Figueroa Nazuno, por creer en sus estudiantes, por sus enseñanzas, ha sido un honor trabajar con usted.*

*Mt 13,3-9*

*A Mario Alberto Angeles Yreta, por su amistad e invaluable integridad que muestra en cada detalle de su vida.*

*Sab 7,14-16*

*A José Luis Martínez, Sergio Bretón y Mónica Nieves, maestros de vida, gracias por su amistad, apoyo y enseñanzas.*

*Col 1,10-14*

*A mis compañeros, Israel Toledo, Iliac Huerta, Miriam Balbuena, Sergio Flores, Sergio Márquez, y Miguel Alejo.*

*Sir 6,18-19*

*Al Instituto Politécnico Nacional*

*Al Centro de Investigación en Computación*

# ÍNDICE

---

<b>RESUMEN / ABSTRACT .....</b>	<b>1</b>
<b>ÍNDICE DE FIGURAS.....</b>	<b>3</b>
<b>ÍNDICE DE TABLAS.....</b>	<b>6</b>
<b>PROLOGO.....</b>	<b>8</b>
a. OBJETIVO.....	8
b. ALCANCES.....	8
c. APORTACIONES.....	10
<b>1. INTRODUCCIÓN.....</b>	<b>12</b>
1.1. El Problema de Similitud.....	13
<b>2. ESTADO DEL ARTE .....</b>	<b>17</b>
2.1. Similitud Entre Códigos Fuente.....	18
<b>3. ALGORITMOS.....</b>	<b>25</b>
3.1. Fast Dynamic Time Warping.....	26
3.1.1. El Warp Path.....	28
3.2. Identificación de Subsecuencias.....	29
3.3. Representando el código fuente en Secuencia.....	37
3.3.1. Codificación Abstracta.....	38
3.3.2. Codificación en Detalle.....	42
<b>4. EL CONJUNTO DE DATOS.....</b>	<b>49</b>
4.1. Creación del conjunto de datos usando códigos fuente .....	50
<b>5. RESULTADOS.....</b>	<b>54</b>
5.1. Comparación de Códigos Fuente .....	55
5.1.1. Codificación Abstracta.....	55

5.1.1.1. Códigos particulares.....	69
5.1.2. Valor de Similitud .....	75
5.1.3. Codificación en Detalle.....	77
5.1.4. Variación de Parámetros.....	83
5.1.4.1. Codificación Abstracta.....	83
5.1.4.2. Codificación en Detalle.....	87
5.1.5. Variación en la Clasificación de operadores.....	89
5.1.6. Comparación de Resultados .....	94
5.1.6.1. Codificación Abstracta .....	94
5.1.6.2. Codificación en Detalle .....	97
5.2. Identificación de Subsecuencias en códigos fuente .....	100
<b>6. CONCLUSIONES.....</b>	<b>108</b>
<b>PUBLICACIONES.....</b>	<b>111</b>
<b>REFERENCIAS.....</b>	<b>113</b>
<b>ANEXOS.....</b>	<b>121</b>
A. Tabla de Resultados, utilizando el valor <i>Sim</i> .	
Clasificación Microsoft. Parámetros $\theta = 1.00$ y $\Delta = 0.05$ . .....	122
B. Tabla de Resultados, utilizando la distancia <i>FDTW</i>	
Clasificación Microsoft. Parámetros $\theta = 1.00$ y $\Delta = 0.05$ . .....	127
C. Otros Experimentos.....	132
D. Código fuente. ....	146

## ÍNDICE DE FIGURAS

---

<b>Figura 1.1</b>	<i>Enfoques del Problema de Semejanza</i>	13
<b>Figura 1.2</b>	<i>Paradigma Clásico del Problema de Semejanza</i>	14
<b>Figura 1.3</b>	<i>Estructura Conceptual del Problema de Semejanza</i>	14
<b>Figura 2.1</b>	<i>Descripción de un código en árbol</i>	18
<b>Figura 2.2</b>	<i>Componentes que integran sentencia IF-THEN-ELSE</i>	19
<b>Figura 2.3</b>	<i>Huella Digital de documento</i>	20
<b>Figura 2.4</b>	<i>Grafo Conceptual de un código en C</i>	21
<b>Figura 2.5</b>	<i>Caracterización de códigos fuente</i>	21
<b>Figura 3.1</b>	<i>Matriz de distancias FDTW.</i>	28
<b>Figura 3.2</b>	<i>Identificación de subsecuencias en matriz FDTW.</i>	30
<b>Figura 3.3</b>	<i>Secuencias Q y C.</i>	31
<b>Figura 3.4</b>	<i>Alineamiento entre secuencias.</i>	31
<b>Figura 3.5</b>	<i>Subsecuencias encontradas entre secuencias</i>	32
<b>Figura 3.6</b>	<i>Cambio de representación propuesto</i>	36
<b>Figura 3.7</b>	<i>Clasificación de operadores por Microsoft.</i>	37
<b>Figura 3.8</b>	<i>Categorías de operadores utilizados</i>	38
<b>Figura 3.9</b>	<i>Valores asignados a los operadores</i>	39
<b>Figura 3.10</b>	<i>Código fuente de entrada, representación en secuencia</i>	40
<b>Figura 3.11</b>	<i>Palabras reservadas</i>	41
<b>Figura 3.12</b>	<i>Representación gráfica de los valores utilizados</i>	42
<b>Figura 3.13</b>	<i>Valores de las palabras reservadas</i>	43
<b>Figura 3.14</b>	<i>Codificación en detalle</i>	44
<b>Figura 3.15</b>	<i>Diagrama de clases</i>	45
<b>Figura 4.1</b>	<i>Código en C# AMP01</i>	50
<b>Figura 4.2</b>	<i>Código en C# AMP11</i>	51
<b>Figura 4.3</b>	<i>Código en C# RAW01</i>	51
<b>Figura 5.1</b>	<i>Códigos Similares a ED01</i>	56

<b>Figura 5.2</b>	<i>Códigos Similares a AMP04</i>	57
<b>Figura 5.3</b>	<i>Códigos Similares a Server02</i>	57
<b>Figura 5.4</b>	<i>Códigos Similares a DMA04</i>	58
<b>Figura 5.5</b>	<i>Código en C# ED01</i>	59
<b>Figura 5.6</b>	<i>Código en C# ED04</i>	60
<b>Figura 5.7</b>	<i>Códigos Similares a Cliente01</i>	62
<b>Figura 5.8</b>	<i>Códigos Similares a HG02</i>	62
<b>Figura 5.9</b>	<i>Representación gráfica de HG01</i>	63
<b>Figura 5.10</b>	<i>Códigos Similares a Rumelhart02</i>	64
<b>Figura 5.11</b>	<i>Códigos Similares a BW01</i>	64
<b>Figura 5.12</b>	<i>Códigos de la clase BW</i>	65
<b>Figura 5.13</b>	<i>Diversos códigos de los experimentos</i>	65
<b>Figura 5.14</b>	<i>Código en C# Cliente01</i>	66
<b>Figura 5.15</b>	<i>Código en C# Cliente02</i>	67
<b>Figura 5.16</b>	<i>Código en C# Euclidean / Euclidean Inverted</i>	68
<b>Figura 5.17</b>	<i>Código en C# HG01</i>	69
<b>Figura 5.18</b>	<i>Código en C# HG01 Moved</i>	70
<b>Figura 5.19</b>	<i>Código en C# Absurdo</i>	71
<b>Figura 5.20</b>	<i>Códigos Similares a Euclidean Inverted</i>	72
<b>Figura 5.21</b>	<i>Códigos Similares a HG01 Moved</i>	73
<b>Figura 5.22</b>	<i>Códigos Similares a Absurdo</i>	73
<b>Figura 5.23</b>	<i>Codificación en detalle RAW01, RAW02</i>	77
<b>Figura 5.24</b>	<i>Codificación en detalle Server02, Server03</i>	78
<b>Figura 5.25</b>	<i>Código en C# Cliente03</i>	79
<b>Figura 5.26</b>	<i>Código en C# Server02</i>	80
<b>Figura 5.27</b>	<i>Código en C# Server03</i>	81
<b>Figura 5.28</b>	<i>Código BW04 con diferentes valores de codificación</i>	83
<b>Figura 5.29</b>	<i>Segunda clasificación de operadores</i>	89
<b>Figura 5.30</b>	<i>Tercera clasificación de operadores</i>	91
<b>Figura 5.31</b>	<i>Subsecuencias identificadas entre AMP03 y AMP09</i>	94



## ÍNDICE DE FIGURAS

---

<b>Figura 5.32</b>	<i>Código AMP03. Subsecuencias identificadas con AMP09</i>	95
<b>Figura 5.33</b>	<i>Código AMP09. Subsecuencias identificadas con AMP03</i>	95
<b>Figura 5.34</b>	<i>Subsecuencias identificadas entre GS01 y GS02</i>	97
<b>Figura 5.35</b>	<i>Código GS01. Subsecuencias identificadas con GS02</i>	98
<b>Figura 5.36</b>	<i>Código GS02. Subsecuencias identificadas con GS01</i>	99

## ÍNDICE DE TABLAS

---

<b>Tabla 4.1</b>	<i>Códigos – Procesamiento de Imágenes</i>	49
<b>Tabla 4.2</b>	<i>Códigos – Sistemas Operativos</i>	50
<b>Tabla 4.3</b>	<i>Códigos – Inteligencia Artificial</i>	50
<b>Tabla 5.1</b>	<i>Tabla de Resultados 1</i>	55 , 84
<b>Tabla 5.2</b>	<i>Tabla de Resultados 2</i>	61
<b>Tabla 5.3</b>	<i>Tabla de Resultados 3</i>	72
<b>Tabla 5.4</b>	<i>Tabla de Resultados 1 utilizando el valor sim</i>	75
<b>Tabla 5.5</b>	<i>Resumen comparativo entre RAW01 y RAW02</i>	76
<b>Tabla 5.6</b>	<i>Resumen comparativo entre Server02 y Server03</i>	78
<b>Tabla 5.7</b>	<i>Tabla de resultados con valores <math>\theta = 2.00</math> y <math>\Delta = 0.50</math></i>	84
<b>Tabla 5.8</b>	<i>Tabla de resultados con valores <math>\theta = 5.00</math> y <math>\Delta = 1.00</math></i>	84
<b>Tabla 5.9</b>	<i>Tabla de resultados con valores <math>\theta = 100.00</math> y <math>\Delta = 7.00</math></i>	85
<b>Tabla 5.10</b>	<i>Resultados codificación en detalle <math>\theta = 1.00</math>, <math>\Delta = 0.05</math></i>	86
<b>Tabla 5.11</b>	<i>Resultados codificación en detalle <math>\theta = 2.00</math>, <math>\Delta = 0.50</math></i>	87
<b>Tabla 5.12</b>	<i>Resultados codificación en detalle <math>\theta = 5.00</math>, <math>\Delta = 1.00</math></i>	87
<b>Tabla 5.13</b>	<i>Resultados codificación en detalle <math>\theta = 100.00</math>, <math>\Delta = 7.00</math></i>	87
<b>Tabla 5.14</b>	<i>Resultados 2ª Clasificación <math>\theta = 1.00</math> y <math>\Delta = 0.05</math>.</i>	90
<b>Tabla 5.15</b>	<i>Resultados 2ª Clasificación <math>\theta = 2.00</math> y <math>\Delta = 0.50</math>.</i>	90
<b>Tabla 5.16</b>	<i>Resultados 3ª Clasificación <math>\theta = 1.00</math> y <math>\Delta = 0.05</math>.</i>	92
<b>Tabla 5.17</b>	<i>Resultados 3ª Clasificación <math>\theta = 2.00</math> y <math>\Delta = 0.50</math>.</i>	92
<b>Tabla 5.18</b>	<i>Comparación Resultados 1ª Clasificación. Abstracta</i>	95
<b>Tabla 5.19</b>	<i>Comparación Resultados 2ª Clasificación. Abstracta</i>	95
<b>Tabla 5.20</b>	<i>Comparación Resultados 3ª Clasificación. Abstracta</i>	95
<b>Tabla 5.21</b>	<i>Comparación Resultados 1ª Clasificación. Abstracta</i>	96
<b>Tabla 5.22</b>	<i>Comparación Resultados 2ª Clasificación. Abstracta</i>	96
<b>Tabla 5.23</b>	<i>Comparación Resultados 3ª Clasificación. Abstracta</i>	97

## ÍNDICE DE FIGURAS

---

<b>Tabla 5.24</b>	<i>Comparación Resultados 1ª Clasificación. Detalle</i>	98
<b>Tabla 5.25</b>	<i>Comparación Resultados 2ª Clasificación. Detalle</i>	98
<b>Tabla 5.26</b>	<i>Comparación Resultados 3ª Clasificación. Detalle</i>	98
<b>Tabla 5.27</b>	<i>Comparación Resultados 1ª Clasificación. Detalle</i>	99
<b>Tabla 5.28</b>	<i>Comparación Resultados 2ª Clasificación. Detalle</i>	99
<b>Tabla 5.29</b>	<i>Comparación Resultados 3ª Clasificación. Detalle</i>	99
<b>Tabla 5.30</b>	<i>Resumen comparativo entre AMP03 y AMP09</i>	100
<b>Tabla 5.31</b>	<i>Resumen comparativo entre GS01 y GS02</i>	103

## **RESUMEN**

*En este trabajo se presenta un nuevo método para el cómputo de la similitud entre códigos fuente de alto nivel. Este método emplea la técnica “Fast Dynamic Time Warping”, que construye un “warp path” o relación entre puntos bajo ciertas restricciones locales, que permite la identificación de instrucciones similares, en base a las clases de operadores que intervienen en un lenguaje de programación. A diferencia de otros métodos de comparación entre códigos fuente, no se realiza una extracción de características. Los resultados obtenidos muestran que dos códigos fuentes serán similares cuando sus secuencias numéricas de representación correspondientes sean similares.*

## ABSTRACT

*This work presents a new approach to compute similarity between high level source codes. The technique is based on Fast Dynamic Time Warping, that builds a warp path or relation of points under local restrictions. The source code is represented into numeric sequences using the operators inside programming languages. This makes possible subsequence detection that represent similar code instructions. In contrast with other code similarity algorithms, we do not make features extraction. The experiments show that two source codes are similar when their respective numeric sequences are similar. A source code is represented as numeric sequences using the operators from the high level language.*

# PRÓLOGO

---

## *OBJETIVO*

El objetivo de este trabajo es, presentar una nueva aproximación al problema de análisis y cómputo de similitud entre códigos fuente de un lenguaje de programación de Alto Nivel; donde se realiza el cambio de representación a las entidades a comparar, mediante su transformación en una *secuencia numérica*; de esta forma, dos códigos fuente serán similares, cuando sus secuencias numéricas de representación correspondientes sean similares.

## *ALCANCES*

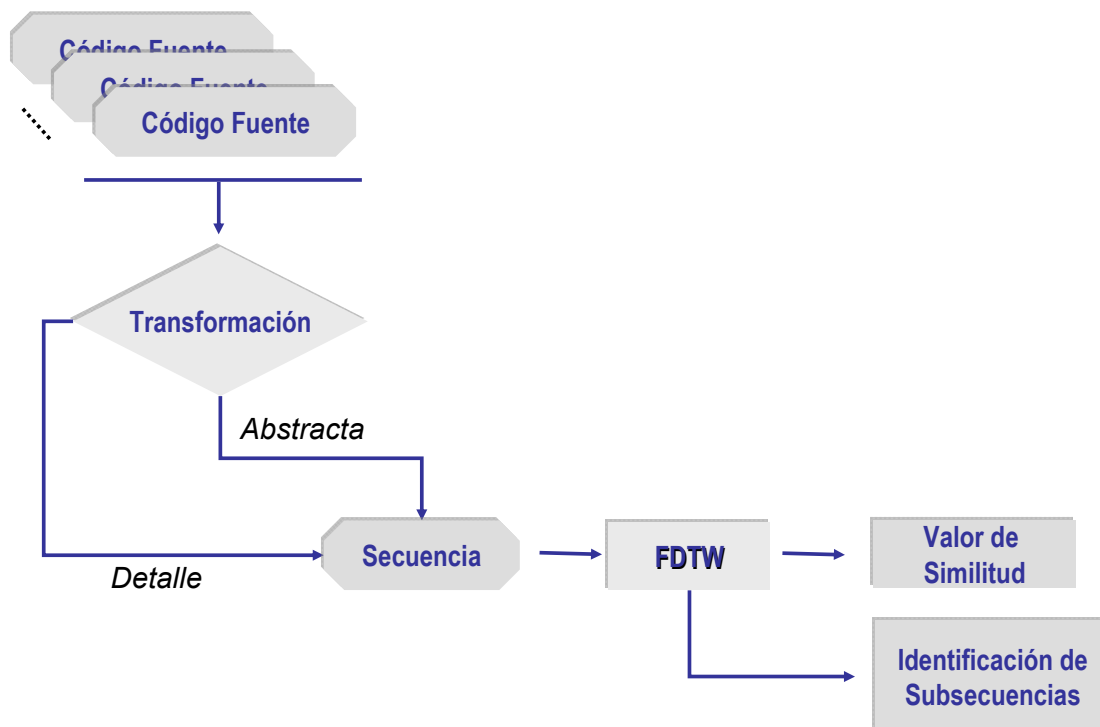
Para lograr este objetivo, se generaron reglas de codificación en base a la clasificación de operadores que intervienen en el lenguaje de programación, obteniendo dos transformaciones:

- i. La primera transformación denominada: *codificación abstracta*, incluye las instrucciones que representan la estructura del código fuente a transformar, identificando de manera única, un conjunto de operadores dentro de una categoría de la clasificación de operadores.
- ii. La segunda transformación denominada: *codificación en detalle*, representa cada elemento que interviene en el código fuente: palabras reservadas, operadores, constantes y variables. Las reglas de codificación, identifican de manera única a cada operador que interviene en el lenguaje de programación.

En este trabajo, también se incluye un algoritmo propuesto para la identificación de *subsecuencias* en secuencias numéricas; el cuál, permite la identificación de instrucciones similares en dos códigos fuente a comparar. Sin embargo, el algoritmo propuesto es generalizable a cualquier fenómeno que pueda ser representado en una secuencia numérica.

Este trabajo **no** incluye una comparación de técnicas para el análisis de similitud entre códigos fuente, debido a que no existe un criterio de comparación universal en el problema de similitud.

Tampoco es un estudio sobre técnicas de extracción/selección de características, esto se debe a que los métodos propuestos no requieren este proceso.



**Figura A.** Metodología aplicada para el análisis de similitud entre códigos fuente.

## ***APORTACIONES***

De forma puntual, las aportaciones de este trabajo son:

- i. Dos nuevas formas de representar códigos fuente de lenguajes de programación de Alto Nivel, en secuencias numéricas. Esto se logra mediante la *codificación abstracta* y la *codificación en detalle*.
- ii. Un nuevo método para la identificación de subsecuencias similares entre dos secuencias numéricas. El método propuesto no incorpora el proceso de extracción de características. Este método se puede aplicar a cualquier fenómeno o entidad que pueda representarse en una secuencia numérica de una dimensión.
- iii. La propuesta de utilizar un nuevo valor de similitud entre códigos fuente de lenguajes de programación de Alto Nivel.
- iv. La sugerencia de realizar cambios de representación de las entidades a comparar, permitiendo el uso de técnicas y métodos de diferentes áreas de especialidad, para realizar el cómputo de similitud.

## ***ESTRUCTURA DEL TRABAJO***

En el capítulo 1, se presenta una introducción al problema de similitud, algunas definiciones y la importancia de este problema.

En el capítulo 2, se presenta el estado del arte, del problema de cómputo de similitud entre códigos fuente.



El capítulo 3, introduce los algoritmos utilizados en este trabajo, el algoritmo *Fast Dynamic Time Warping*, para el cómputo de similitud entre secuencias numéricas. El algoritmo propuesto para la identificación de subsecuencias; los algoritmos para la transformación del código fuente en una *secuencia numérica*. Los algoritmos presentados en este trabajo, son de bajo costo computacional, esto es, la *complejidad espacial y temporal*, son *polinomiales*.

En el capítulo 4, se describe el conjunto de datos utilizado para los experimentos.

El capítulo 5, abarca los resultados obtenidos para la comparación de códigos fuente utilizando la codificación abstracta y en detalle; se propone un valor de similitud para la comparación de códigos fuente; además, se muestran resultados utilizando diferentes clasificaciones de operadores para la codificación. Este capítulo incluye, los resultados para la identificación de instrucciones similares en códigos fuente.

Por último, en el capítulo 6, se incluyen las conclusiones derivadas de este trabajo.

# **CAPÍTULO 1**

---

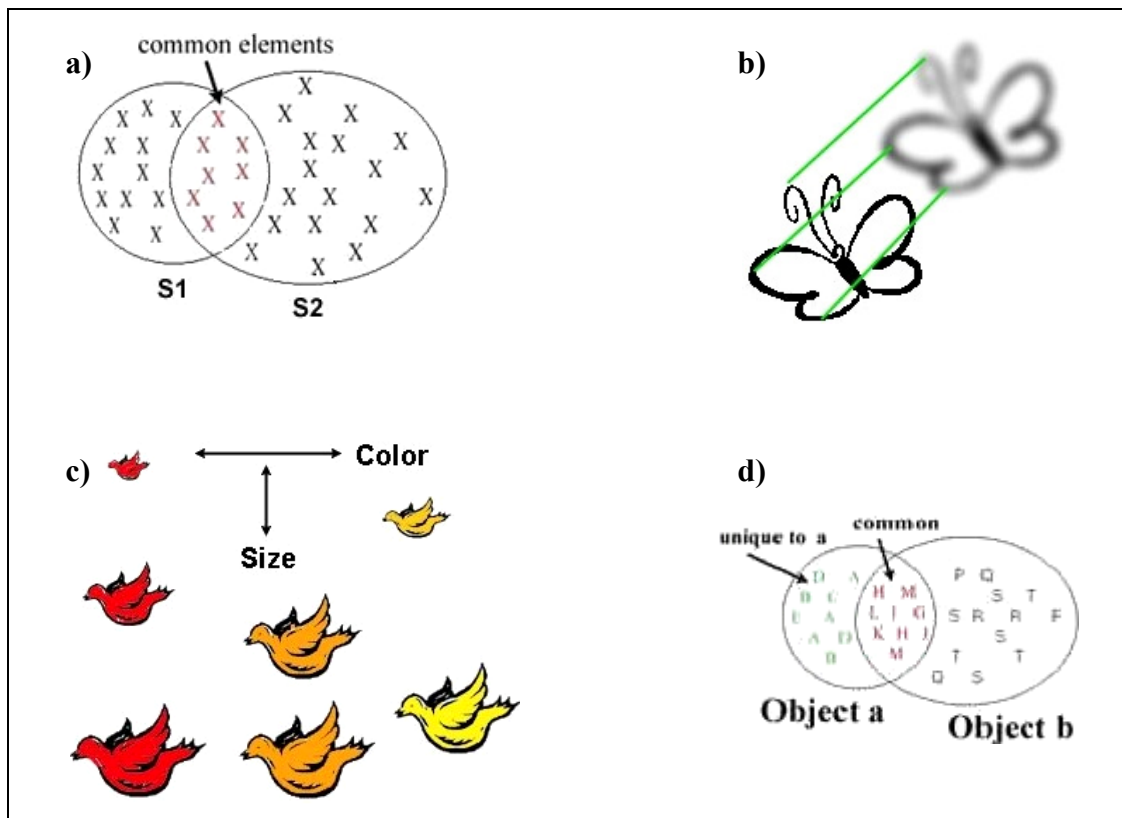
## **INTRODUCCIÓN**

## El Problema de Similitud

Los primeros esfuerzos para entender el problema de semejanza los dió Tversky [Tversky77] en 1977, cuando definió que el problema de semejanza se puede abordar en cuatro enfoques distintos, ver la Figura 1.1. El primer enfoque (Figura. 1.1a) se refiere a la comparación de entidades usando como criterio los *elementos comunes* (*common elements approach*). Otro enfoque (Figura 1.1b) es el de realizar comparaciones en donde se persigue *emparejar* o *alinear* una entidad con respecto a la otra (*template approach*). Un tercer enfoque (Figura 1.1c) se refiere a las comparaciones de entidades por su *geometría* (*geometric approach*). Por último en (Figura 1.1d) se muestra el enfoque de *características* (*feature approach*) que es similar al de *elementos comunes*, aunque este último no requiere una identificación *a priori* de las propiedades involucradas en la comparación.

### Estructura del Problema de Similitud

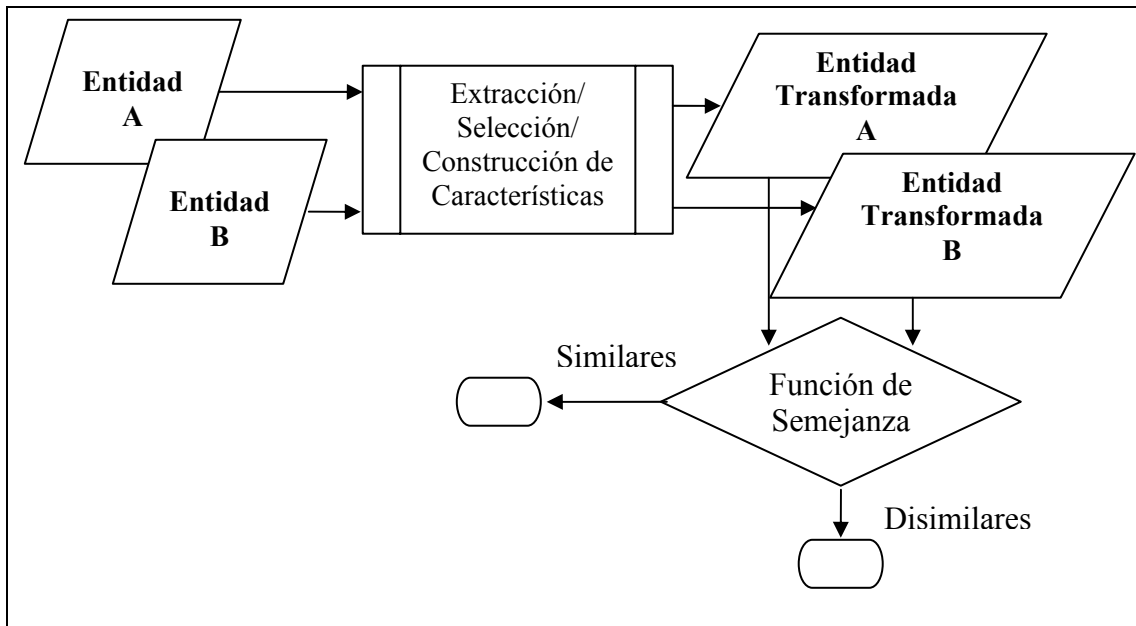
Para realizar el análisis de semejanza entre dos entidades, es necesario que estas entidades sean *cuantificables*, esto es, asignar un valor a las entidades siguiendo un conjunto de reglas.



**Figura. 1.1** Enfoques del problema de semejanza declarados por Tversky, imágenes extraídas de [Tversky77].

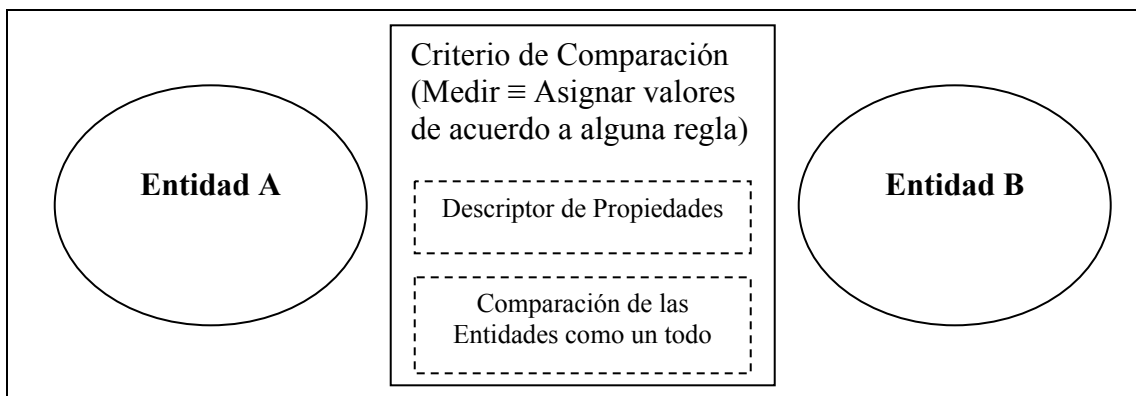
En [Figuroa-Nazuno06], se presenta el paradigma clásico y el paradigma moderno del problema de similitud. Figura 1.2 y Figura 1.3 respectivamente.

En el paradigma clásico, se incluyen etapas que abarcan los procesos de pre-procesamiento, extracción / selección de características, normalización,..., entre otros procesos, donde se requiere conocer *a priori*, las características descriptivas de las entidades a comparar, dejando de lado el resto de la información.



**Figura 1.2.** El *paradigma* clásico para el problema de computar semejanza entre entidades.

Por otro lado, el paradigma moderno, establece los procesos previos a la función de semejanza del paradigma clásico como opcionales. Permitiendo comparar las entidades como un todo, donde el criterio de comparación es un proceso para asignar números en base a alguna regla. Ver Figura 1.3.



**Figura 1.3** Estructura conceptual del problema de semejanza. Las entidades a comparar deben ser cuantificables.

Este trabajo se encuentra organizado de la siguiente forma: en el Capítulo 3, se describen las técnicas y el procedimiento utilizado para la comparación de códigos fuente. Posteriormente, en el Capítulo 4, se describen los conjuntos de datos sobre los que se realizaron los experimentos de este trabajo. En el capítulo 5, se muestran los resultados obtenidos, utilizando códigos fuentes que corresponden a asignaciones en clase de programación, del Instituto Politécnico Nacional. En el Capítulo 6, se presentan las Conclusiones. Posteriormente, se encuentra un listado de las publicaciones relacionadas con este trabajo, y aquellas que surgieron en paralelo con esta tesis. Finalmente se muestran las Referencias y Anexos.

## Referencias del capítulo

[Tversky77] Tversky A., Shafir E. “*Preference, Belief, and Similarity*”. Selected Writings, Ed. MIT Press, Cambridge, MA, 2004. Paperback: 1039 pp., illus. ISBN 026270093X.

[Figuroa-Nazuno06] J. Figuroa-Nazuno, A. Angeles-Yreta, J. Medina-Apodaca, V. Ortega-González, K. Ramírez-Amaro, M. Mirón-Bernal, V. Landassuri-Moreno. “*Sobre el problema de Semejanza*”. Reporte Técnico. Centro de Investigación en Computación. Instituto Politécnico Nacional. Unidad Profesional “Adolfo López Mateos”. ISBN: 970-36-0343-2. México, DF. 2006.

[Angeles06] A. Angeles-Yreta. “*Cómputo de la Similitud entre Figuras Geométricas*”. Tesis de Maestría en Ciencias de la Computación del Centro de Investigación en Computación del Instituto Politécnico Nacional, 2006.

## **CAPÍTULO 2**

---

### **ESTADO DEL ARTE**

## Similitud entre Códigos Fuente

Cuando los programas de cómputo se encuentran en proceso de desarrollo, comúnmente los programadores utilizan rutinas de código duplicadas, y con frecuencia, estas rutinas de código son adaptadas. En algunos casos, copiando la parte sustancial de una idea para apropiarse de ella; esto es conocido como plagio o código reusable.

El realizar copias de fragmentos de código, es un ejercicio habitual en las instituciones educativas que cuentan con grupos de enseñanza de programación de computadoras. Algunos métodos frecuentes al realizar esta práctica son: realizar cambios en los nombres de variables, tipos de datos, comentarios; intercambiar la secuencia de las instrucciones; para esto, se requiere un nivel de entendimiento del código a ser copiado, que en algunos casos, debido a la presión por “*liberar*” el programa, no se cuenta con el tiempo para analizarlo detalladamente.

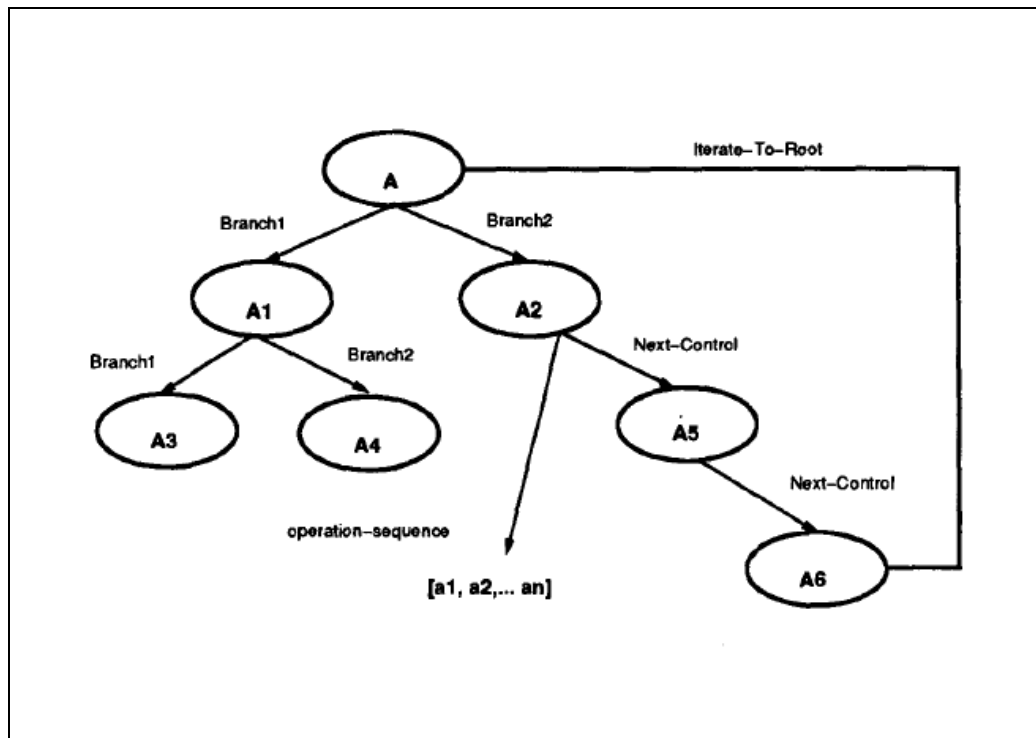
Aun al realizar estas modificaciones, se preserva la estructura y la *esencia* del código fuente original, de otra forma, no tendría sentido modificarlo ya que probablemente no realizaría la misma función.

Para obtener la estructura mencionada anteriormente, se realiza una transformación sobre el código fuente, eliminando los comentarios, las dependencias entre códigos (*headers*), reemplazando los nombres de variables y literales. Con ello se obtiene una representación a nivel de instrucción, esto es, identificar operaciones como son de: *asignación, aritméticas, lógicas, relacionales, incremento/decremento, indexado,*



llamadas a función, declaraciones, entre otras. Este procedimiento se encuentra detallado en el Capítulo 3.

En [Kontogiannis93], se muestra una aproximación al problema de calcular la similitud entre códigos fuente. Kontogiannis propone realizar un cambio de representación de los códigos, y presentarlos en forma de una estructura computacional denominada árbol. Ver Figura 2.1.



**Figura 2.1** Ejemplo de la descripción de un programa en un árbol. Tomado de [Kontogiannis93].

Sin embargo, este método resulta poco eficiente, ya que los códigos fuente de Alto Nivel, permiten anidaciones de más de tres niveles, provocando que la profundidad del árbol se incremente desmesuradamente. En la Figura 2.2, se muestra la representación de la sentencia IF-THEN-ELSE, comúnmente utilizada en los lenguajes de programación. Ver [Baker98] [Sager06]

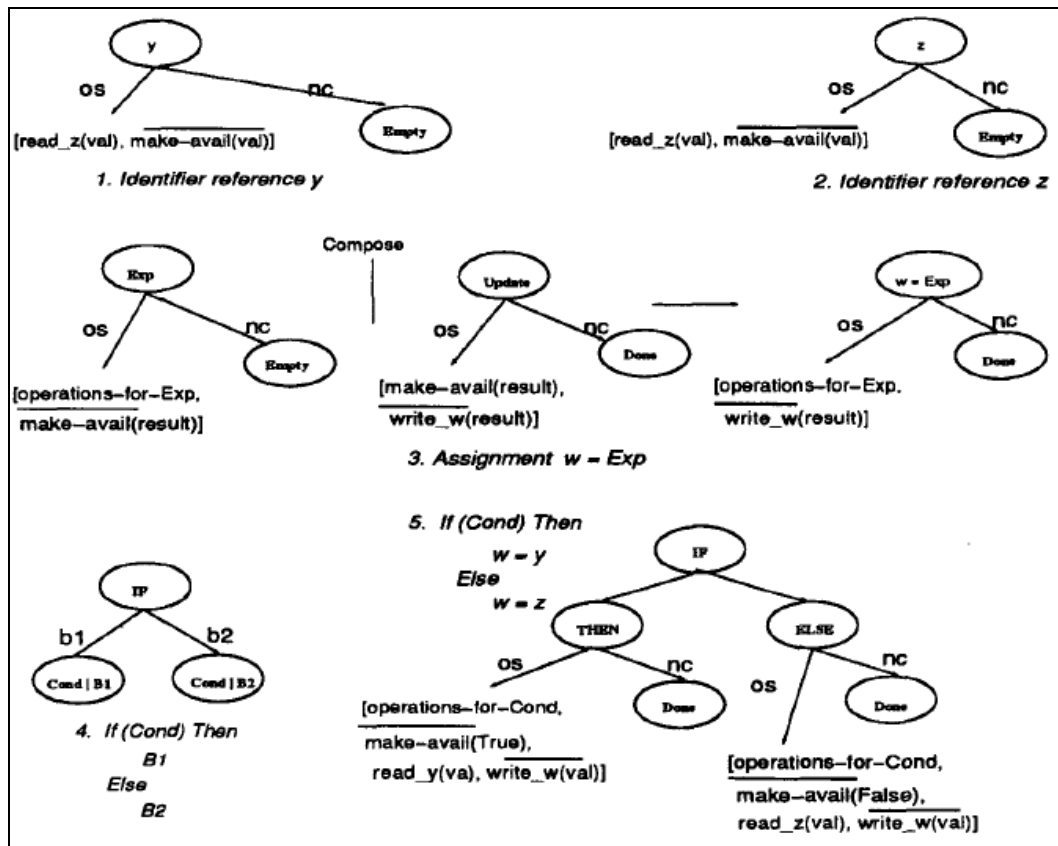


Figura 2.2. Componentes que integran una sentencia IF-THEN-ELSE. Tomado de [Kontogiannis93].

En el mismo año, Schleimer [Schleimer03] en un trabajo conjunto con la Universidad de Berkeley, presenta un algoritmo denominado *Winnowing*. Este algoritmo se encuentra intrínseco en la herramienta *MOSS*, que se convertiría en el programa de detección de plagio entre códigos fuente más conocido y utilizado por las Universidades de Estados Unidos. En la actualidad, este programa puede ser accedido vía Internet, mediante un proceso sencillo de registro.

El algoritmo *Winnowing*, realiza comparación entre secuencias numéricas utilizando gramáticas. Este algoritmo remueve características irrelevantes del código, para realizar el desplazamiento de una ventana de tamaño  $k$  sobre el texto; de esta ventana se obtienen tablas de *hash* para representar el código en secuencia. Una vez realizado

este paso, se obtiene un gramática para esta secuencia de números y finalmente obtener una *huella digital* del documento a comparar. Ver Figura 2.3. Una nota en este artículo es : si dos secuencias *hash* son similares, no se garantiza que lo sean sus documentos correspondientes. En el mismo artículo, los autores presentan un algoritmo donde aseguran, la existencia de similitud entre los textos representados en una secuencia *hash*.

```

A do run run run, a do run run
(a) Some text from [7].

adorunrunrunadorunrun
(b) The text with irrelevant features removed.

adoru dorun orunr runru unrun nrunr runru
unrun nruna runad unado nador adoru dorun
orunr runru unrun
(c) The sequence of 5-grams derived from the text.

77 72 42 17 98 50 17 98 8 88 67 39 77 72 42
17 98
(d) A hypothetical sequence of hashes of the 5-grams.

72 8 88 72
(e) The sequence of hashes selected using  $0 \text{ mod } 4$ .

```

**Figura 2.3** Huella Digital de un documento utilizando el algoritmo *Winning*. Imagen tomada de [Schleimer03].

En [Mishne04], se muestra otra aproximación al cómputo de similitud entre códigos fuente, donde se utilizan grafos conceptuales. Para realizar esta representación, es necesario un etiquetado, que presenta un problema similar al presentado en [Kontogiannis93]. Ver Figura 2.4. El lector puede percibir, la cantidad de etiquetas que se requiere para un código sencillo.

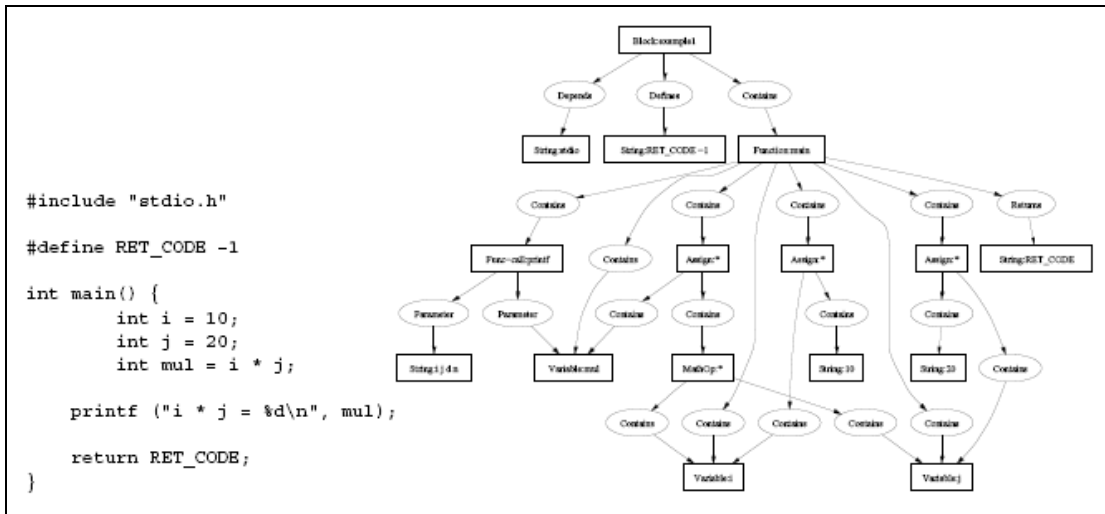


Figura 2.4 Grafo conceptual de un código en C.

En algunos casos, han existido aproximaciones al problema de computar semejanza entre códigos fuentes utilizando vectores de características. [Basit05] [Mann06] Esto representa un problema: el elegir las características *representativas* del código fuente. Estos trabajos incluyen características como: número de líneas (LOC), complejidad ciclométrica (CC), número de funciones (FN), líneas de comentarios (CN), y la métrica *Free Text Similarity* (FT), entre otros. Ver Figura 2.5.

04_1	LOC	CC	FN	CN	FT
Average	350.86	86.07	72.14	56.64	59.96
Stdev	98.44	27.80	25.94	47.55	31.00
Max	563	145	131	193	100
Min	116	18	10	0	0
Stdev % of Avg	28%	32%	36%	84%	52%
04_2	LOC	CC	FN	CN	FT
Average	228.19	47.95	23.86	60.43	13.14
Stdev	140.30	31.46	12.83	35.95	21.00
Max	797	179	72	177	79
Min	123	25	11	12	0
Stdev % of Avg	61%	66%	54%	59%	160%

Figura 2.5 Caracterización de códigos fuente. Tomado de [Mann06]

En [Arwin06], se presenta el cómputo de semejanza entre códigos fuente utilizando el código intermedio generado por los compiladores de Alto Nivel.

Por otro lado, esta tesis abarca la comparación de códigos fuente de alto nivel y la identificación de subsecuencias similares entre ellos. Sin embargo, el método para identificar subsecuencias, propuesto en esta tesis, tiene como base obtener una aproximación al problema publicado en [Hirschber75]. Donde se presenta un algoritmo como propuesta al problema denominado “*The Longest Common Subsequence*” (*LCS*), el cual es un problema clásico de la Ciencia de la Computación y además es un problema NP-Completo.

## Referencias del capítulo

[Figuroa-Nazuno06] J. Figuroa-Nazuno, A. Angeles-Yreta, J. Medina-Apodaca, V. Ortega-González, K. Ramírez-Amaro, M. Mirón-Bernal, V. Landassuri-Moreno. “*Sobre el problema de Semejanza*”. Reporte Técnico. Centro de Investigación en Computación. Instituto Politécnico Nacional. Unidad Profesional “Adolfo López Mateos”. ISBN: 970-36-0343-2. México, DF. 2006.

[Angeles06] A. Angeles-Yreta. “*Cómputo de la Similitud entre Figuras Geométricas*”. Tesis de Maestría en Ciencias de la Computación del Centro de Investigación en Computación del Instituto Politécnico Nacional, 2006.

[Kontogiannis93] Kostas Kontogiannis. “*Program representation and behavioural matching for localizing similar code fragments*”. Proceedings of the 1993 conference of the Center for Advanced Studies on Collaborative research: software engineering. Volume 1, Pages 194 – 205, 1993.

[Baker98] Brenda S. Baker, Raffaele Giancarlo. “*Longest Common Subsequence from Fragments via Sparse Dynamic Programming*”. Proceedings of the 6th Annual European Symposium on Algorithms pp. 79 – 90, 1998.

[Schleimer03] Saul Schleimer, Daniel S. Wilkerson, Alex Aiken. “*Winnowing: local algorithms for document fingerprint*”. International Conference on Management of Data. Proceedings of the 2003 ACM SIGMOD international conference on Management of Data. San Diego, California, pp. 76 - 85 , 2003 .

[Mishne04] G. Mishne and M. de Rijke. “*Source Code Retrieval using Conceptual Similarity*”. Proceedings of the *Recherche d’Information Assiteé par Ordinateur*. 2004.

[Basit05] Hamid Abdul Basit, Stan Jarzabek. “*Detecting higher-level similarity patterns in programs*”, Proceedings of the 10th European software engineering conference. ACM SIGSOFT International Symposium on Foundations of software Engineering, pp. 156 - 165 , 2005

[Mann06] Samuel Mann, Zeldia Frew. “*Similarity and originality in code: plagiarism and normal variation in student assignments*”. Proceedings of the 8th Australian conference on Computing education . pp. 143 - 150, 2006.

[Arwin06] Christian Arwin, S. M. M. Tahaghoghi. “*Plagiarism detection across programming languages*”. Proceedings of the 29th Australasian Computer Science Conference – Volume. pp. 277 - 286 , 2006 .

[Hirschberg75] D. S. Hirschberg. “*A linear space algorithm for computing maximal common subsequences*”. Communications of the ACM Volume 18, Issue 6 (June 1975), pp. 341 – 343, 1975 .

# **CAPÍTULO 3**

---

## **ALGORITMOS**

## Fast Dynamic Time Warping

*Fast Dynamic Time Warping (FDTW)*, es un algoritmo basado en los principios de la programación dinámica<sup>1</sup>, que minimiza una distancia base (3.3) [comúnmente es la distancia euclideana] entre dos secuencias numéricas (3.2)  $Q$  y  $C$ , de tamaño  $|Q|$ ,  $|C|$ <sup>2</sup> respectivamente, esto es:

$$FDTW(Q, C) \leq D_{base}. \quad 3.1$$

También se dice que realiza un “alineamiento” temporal entre las dos secuencias, es decir, construye una relación de puntos denominados *warp path*, donde el  $i$ -ésimo punto de  $Q$ , es relacionado con el  $j$ -ésimo punto de  $C$ . Este algoritmo fue mostrado en [Jang00].

Antes de mostrar las propiedades de *FDTW*, y su estructura son necesarias algunas definiciones.

**Definición 3.1:** *Secuencia.* Conjunto de valores ordenados que pertenecen al conjunto de los números reales.

$$Q = \{ q_1, q_2, q_3 \dots q_n \}, \forall q \in \mathbf{R} \quad 3.2$$

---

<sup>1</sup> Es un concepto matemático para el análisis de procesos que involucra conceptos de optimalidad. [Silverman&Morgan 90]

<sup>2</sup>  $| \cdot |$  Denota el número de elementos contenidos en una secuencia.



**Definición 3.2:** *Distancia Euclideana.*

Dadas dos secuencias  $Q = \{q_1, \dots, q_i, \dots, q_n\}$  y  $C = \{c_1, \dots, c_j, \dots, c_n\}$ , la distancia euclidiana  $D_{euc}$  se define como:

$$D_{euc}(Q, C) = \sqrt{\sum_{i=1}^n (q_i - c_i)^2} \quad 3.3$$

Una vez realizadas estas definiciones, se presentan las propiedades de *FDTW*:

$$FDTW(\langle \rangle, \langle \rangle) = 0 \quad 3.4$$

$$FDTW(Q, \langle \rangle) = FDTW(\langle \rangle, C) = \infty \quad 3.5$$

$$D_{base}(q_i, c_j) = \sqrt{(q_i - c_j)^2} \quad 3.6$$

La distancia *FDTW* se obtiene de la siguiente forma:

<b>Algoritmo FDTW (Q,C)</b>	
	<i>//Calcular el primer elemento</i>
1.	$M[0,0] = D_{base}(q_0, c_0)$
	<i>//Calcular la primera fila</i>
2.	$M[0,j] = M[0,j-1] + D_{base}(q_0, c_j)$
	<i>//Calcular la primera columna</i>
3.	$M[i,0] = M[i-1,0] + D_{base}(q_i, c_0)$
	<i>//Calcular el resto de la matriz</i>
4.	$M[i,j] = D_{base}(q_i, c_j) + \min\{M[i-1,j-1], M[i-1,j], M[i,j-1]\}$

De esta forma se dice que:

$$FDTW(Q, C) \rightarrow \infty \quad \text{si } Q \text{ y } C \text{ son diferentes} \quad 3.7$$

$$FDTW(Q, C) = 0 \quad \text{si } Q \text{ y } C \text{ son iguales} \quad 3.8$$

$$FDTW(Q, C) = FDTW(C, Q) \quad 3.9$$

Como se puede observar en (3.7), la distancia *FDTW* es un valor escalar que pertenece al conjunto de los números reales.

## El Warp Path

El *warp path* ( $W$ ), es un arreglo bidimensional, que contiene la relación de puntos (mapeo) entre las secuencias  $Q$  y  $C$ . Donde cada  $k$ -ésimo elemento de  $W$  está definido como:

$$W = \{ w_1, w_2, w_k, \dots, w_K \} \quad 3.10$$

$$\max(|Q|, |C|) \leq K \leq (|Q| + |C|) - 1 \quad 3.11$$

$$w_k = (i, j) \text{ con } 1 \leq i \leq |Q| \text{ y } 1 \leq j \leq |C| \quad 3.12$$

El *warp path* está sujeto a las siguientes restricciones:

1. **Inicio / Fin:** Requiere que el elemento  $q_1$  sea relacionado con  $c_1$ , y el elemento  $q_n$  sea relacionado con  $c_m$ .
2. **Monotonicidad:** Los elementos del *warp path* deben cumplir con la siguiente condición:  $w_{k-1} \leq w_k$  en sus índices  $(i, j)$ .
3. **Restricciones Locales:** Las restricciones locales se utilizan para reducir el rango de búsqueda de los elementos de  $w$  en base a la vecindad de un elemento en la matriz.

En la Figura 3.1, se muestra un ejemplo del cálculo de *FDTW* entre dos secuencias y el *warp path* que le corresponde. Para este ejemplo la distancia *FDTW* (Secuencia  $C$ , Secuencia  $Q$ ) = 3. A continuación se muestra el *warp path* de la Figura 3.1.

	<i>Elementos (Índices)</i>												
<b>Secuencia Q</b>	1	2	3	4	5	6	7	8	9	9	9	9	9
<b>Secuencia C</b>	1	1	1	2	2	2	3	4	5	6	7	8	9

		1	1.5	3	2	1	1	1	0	0	
1	↖	0	0.5	2.5	3.5	3.5	3.5	3.5	4.5	5.5	
1	0	↖	0.5	2.5	3.5	3.5	3.5	3.5	4.5	5.5	
1	0	0.5	↖	2.5	3.5	3.5	3.5	3.5	4.5	5.5	
1.5	0.5	0	1.5	↖	2	2.5	3	3.5	5	6	
1.5	1	0	1.5	2	↖	2.5	3	3.5	5	6.5	
2	2	0.5	1	1	2	↖	3	4	5.5	7	
3	4	2	0.5	1.5	3	4	↖	5	7	8.5	
1.5	4.5	2	2	1	1.5	2	2.5	↖	4	5.5	
1	4.5	2.5	4	2	1	1	1	2	↖	<b>3</b>	
											Secuencia Q
											Secuencia C

**Figura 3.1.** Matriz de distancias FDTW. En el marco con bordes en rojo se muestra el warp path calculado, y en la posición 9,9 de la matriz, el coste del camino igual a 3.

La complejidad espacial de *FDTW* es de  $|Q| \times |C|$ , la complejidad temporal está definida por  $O(|C-I|) + O(|Q-I|) + O(|C-I|*|Q-I|)$ .

## Identificación de Subsecuencias

**Definición 3.3.** *Subsecuencia.* Es un conjunto de elementos contiguos que pertenecen a una secuencia  $Q$ , usando la siguiente definición:

$$sub(Q) = \{s_1, s_2, s_3 \dots s_z\}, \forall s \in Q \tag{3.13}$$

En la Figura 3.1, se muestra la comparación de dos secuencias obteniendo  $FDTW(Q,C)=3$ ; se muestran los elementos que conforman el *warp path* resaltados con borde grueso. La diagonal principal de la matriz es marcada por el símbolo ↖;

cuando los elementos que pertenecen a  $W$  son iguales a los elementos que componen la diagonal principal se obtiene  $FDTW(Q,C)=0$ , esto es, las secuencias de entrada son idénticas.

Es aquí donde se basa la **idea propuesta** de identificar subsecuencias:

*El grado de similitud de las subsecuencias, está basada en la relación de los elementos de  $W$  con la diagonal principal. Aquellos elementos que se encuentren dentro de  $W$  y que sean “paralelos” a la diagonal principal, conservarán un grado de semejanza; mientras más cerca de la diagonal principal se encuentren, mayor será la similitud entre ellos.*

A continuación se presenta el pseudo código de la idea presentada:

**Algoritmo: Identificación de subsecuencias.**

1. Obtener el warp path  $\mathbf{W}$  de BackTracking
2. Desde  $\mathbf{W}_1$  hasta  $\mathbf{W}_K$  (Ver Ec. 3.11)
  - 2.1. Si  $w_k(i, j)$  es igual con  $w_{k+1}(i-1, j-1)$ 
    - 2.1.1. Marcar  $w_k$  y  $w_{k+1}$  como subsecuencia
3. Fin Desde

**Algoritmo: BackTracking**

1. Obtener la matriz de distancias  $\mathbf{M}$  de  $FDTW$
2. Agregar al warp path  $M[|Q|, |C|]$
3. Recorrer la matriz  $M$ , mientras no se agregue la celda  $M[1, 1]$  a  $\mathbf{W}$  (Ver Ec. 3.10)
  - 3.1. Agregar al warp path los índices de  $\min\{M[i-1, j-1], M[i-1, j], M[i, j-1]\}$
4. Agregar al warp path  $M[1,1]$

En el ejemplo de la Figura 3.1 obtuvimos el warp path:

*Elementos (Índices)*

<b>Secuencia Q</b>	1	2	3	4	5	6	7	8	9	9	9	9	9
<b>Secuencia C</b>	1	1	1	2	2	2	3	4	5	6	7	8	9

Al aplicar el algoritmo propuesto anteriormente identificamos los elementos del *warp path* que corresponden a una subsecuencia. [Elementos sombreados].

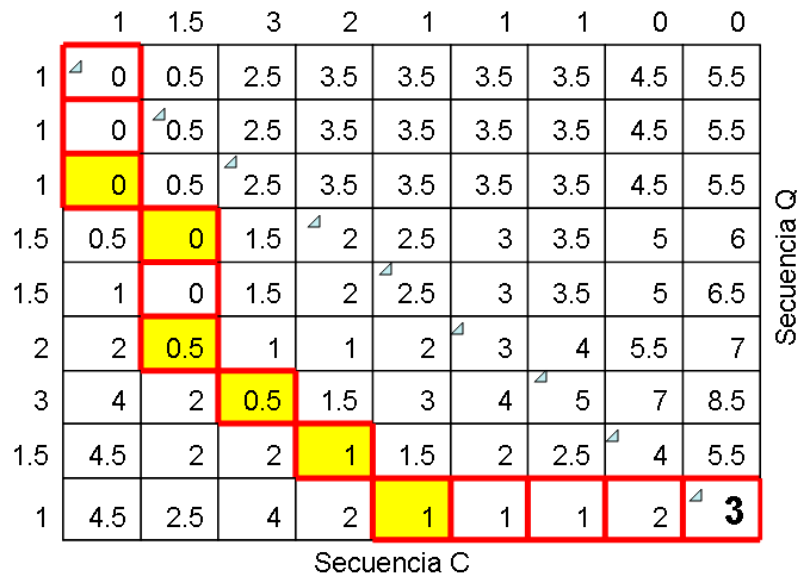
*Elementos (Índices)*

<b>Secuencia Q</b>	1	2	3	4	5	6	7	8	9	9	9	9	9
<b>Secuencia C</b>	1	1	1	2	2	2	3	4	5	6	7	8	9

Al obtener los elementos referenciados por el *warp path* obtenemos **dos subsecuencias similares**.

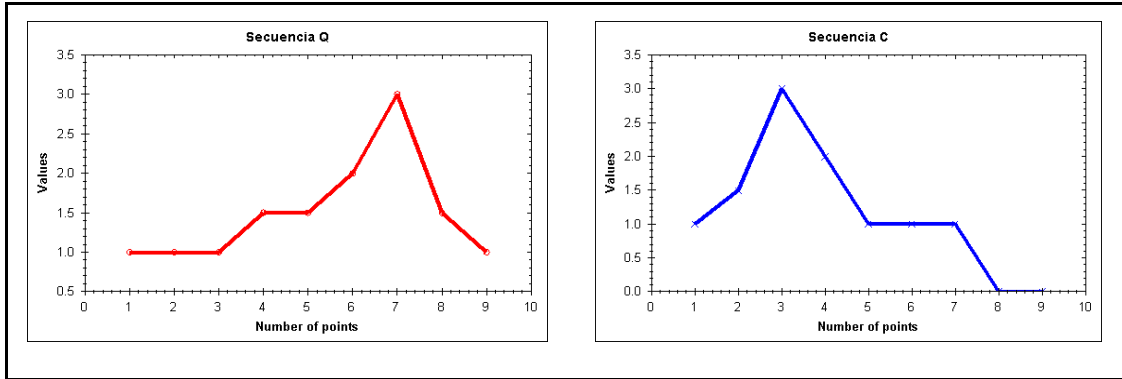
$$S1 : Q\{1, 1.5\} \text{ y } C\{1, 1.5\}$$

$$S2 : Q\{2, 3, 1.5, 1\} \text{ y } C\{1.5, 3, 2, 1\}$$

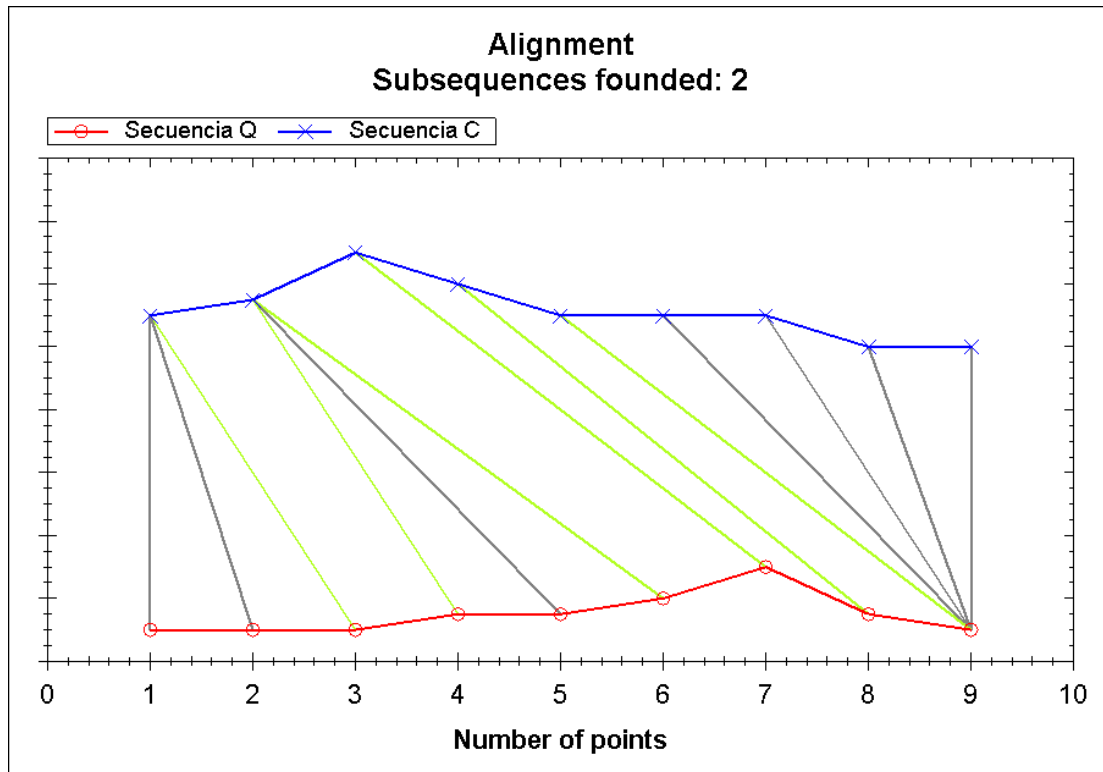


**Figura 3.2.** Identificación de subsecuencias. En el marco con bordes en rojo se muestra el warp path calculado, los elementos sombreados pertenecen a una subsecuencia.

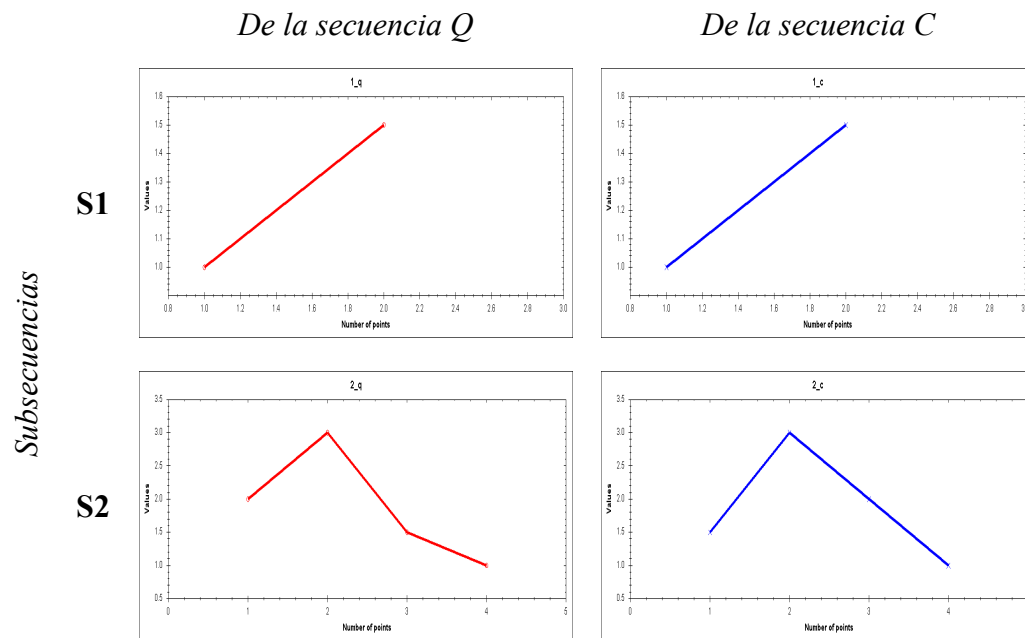
A continuación se muestran las representaciones gráficas de las secuencias  $Q$  y  $C$  respectivamente. Figurar 3.3. El alineamiento producido por  $FDTW$  en el  $warp\ path$  Figura 3.4. y las subsecuencias obtenidas. Figura 3.5.



**Figura 3.3.** Representación gráfica de las secuencias  $Q$  (rojo) y  $C$  (azul) del ejemplo mostrado en la Figura 3.2



**Figura 3.4.** Alineamiento entre secuencias. Las subsecuencias son mostradas en color verde. El alineamiento producido por el  $warp\ path$  son las líneas en color negro.



**Figura 3.5.** Subsecuencias encontradas mediante la técnica propuesta.

La técnica propuesta ha sido mostrada con Series de Tiempo, [Mirón-Bernal06] obteniendo los resultados esperados, identificar subsecuencias comunes a ambas Series de Tiempo, utilizando un **algoritmo no supervisado, automático, y sin realizar extracción de características**.

La complejidad espacial del algoritmo de identificación de subsecuencias es  $(|Q|+|C|) - 1$  y la complejidad temporal es  $O(4[|Q|+|C|] - 1)$ .

A continuación se muestran los códigos de *FDTW*, *Backtracking* e *Identificación de subsecuencias* programados en C# de .NET.

**Método:** *GetDistance()*, pertenece a la clase *FDTW*.

**Entradas:** *Secuencias Q, C*

**Salida:** *Distancia Fast Dynamic Time Warping. Valor de Similitud.*

```
public double GetDistance()
{
    //Crear la matriz de distancias
    M = new double[Q.Length, C.Length];

    //Calcular la distancia base entre los primeros elementos
    M[0, 0] = euclidean(Q[0], C[0]);

    //Calcular las distancias para la primera fila
    for (int j = 1; j < C.Length; j++)
        M[0, j] = M[0, j - 1] + euclidean(Q[0], C[j]);

    //Calcular las distancias para la primera columna
    for (int i = 1; i < Q.Length; i++)
        M[i, 0] = M[i - 1, 0] + euclidean(Q[i], C[0]);

    //Calcular las distancias para el resto de la matriz
    for (int i = 1; i < Q.Length; i++)
        for (int j = 1; j < C.Length; j++)
        {
            double distance = euclidean(Q[i], C[j]);

            M[i, j] = M[i - 1, j - 1] + distance;

            if ((M[i - 1, j] + distance) < M[i, j])
                M[i, j] = M[i - 1, j] + distance;

            if ((M[i, j - 1] + distance) < M[i, j])
                M[i, j] = M[i, j - 1] + distance;
        }

    //retornar la distancia FDTW
    return M[Q.Length - 1, C.Length - 1];
}
```

**Método:** *Euclidean()*, pertenece a la clase *FDTW*.

**Entradas:** *Valores  $q_i$ ,  $c_j$ , miembros de Q y C, respectivamente.*

**Salida:** *Valor de la distancia base, en este caso, distancia euclideana.*

```
private double euclidean(double q, double c)
{
    return (Math.Sqrt(Math.Pow(q - c, 2)));
}
```



**Método:** *BackTracking()*, pertenece a la clase *FDTW*.

**Entradas:** *Matriz de distancias calculadas por el método GetDistance()*

**Salidas:** *Warp path*

```
private void BackTracking()
{
    double distance = 0.00;
    int i, j;           //Indices para la matriz de distancias
    node p = new node(); //Contiene un elemento del warp path

    ArrayList path = new ArrayList(); //Arreglo temporal

    p.q = i = Q.Length - 1; //Warp path siempre incluye los
    p.c = j = C.Length - 1; //elementos de la matriz [Q,C]

    path.Add(p); //Agregar el nodo p al warp path

    //Recorrer la matriz hasta alcanzar la primera fila o la primera
    //columna M[0,1] or M[1,0]

    while (i > 1 || j > 1)
    {
        //Estructura que contiene los índices del warp path
        p = new node();

        //Buscar la distancia mínima de los vecinos de M[I,j] a 45,0,
        //y 90 grados.

        //Vecino 45 Grados
        if (i >= 1 && j >= 1)
        {
            distance = M[i - 1, j - 1];
            p.q = i - 1;
            p.c = j - 1;
        }
        else
            distance = Double.MaxValue;

        //0 Grados
        if (i >= 0 && j > 0)
            if (M[i, j - 1] < distance)
            {
                p.q = i;
                p.c = j - 1;
                distance = M[i, j - 1];
            }

        //90 Grados
        if (i > 0 && j >= 0)
            if (M[i - 1, j] < distance)
            {
                p.q = i - 1;
                p.c = j;
            }
    }
}
```

```

    }

    i = p.q;
    j = p.c;

    //Agregar al path el vecino con la distancia mínima
    path.Add(p);
}

//Siempre incluye los elementos 0,0
p = new node();
p.q = p.c = 0;
path.Add(p);

//Invertir el orden del path
path.Reverse();
Path = (node[])path.ToArray(typeof(node));
}

```

**Método:** *IdentificacióndeSubsecuencias()*, pertenece a la clase *FDTW*.

**Entradas:** *Warp path del método BackTracking*

**Salidas:** *Subsecuencias identificadas en el warp path*

```

private void IdentificaciondeSubsecuencias ()
{
    int nsub = 0; bool found = false;

    for (int z = 0; z < Path.Length -1; z++)
        if (Math.Abs(Path[z].q - Path[z + 1].q) == 1 &&
            Math.Abs(Path[z].c - Path[z + 1].c) == 1)
            { //Marcar como subsecuencia
                Path[z].IsSub = true;
                Path[z].nsub = nsub;

                Path[z + 1].IsSub = true;
                Path[z + 1].nsub = nsub;
                found = true;
            }
        else
            if (found)
                {
                    nsub++; found = false;
                }

    patterns_founded = nsub + 1;
}

```

# Representando El Código Fuente En Secuencia

Ahora explicaremos la transformación de representación aplicada a los códigos fuentes para conseguir su representación en una secuencia (3.2). Para lograr representar el código fuente de entrada en una secuencia numérica, se utilizan los operadores que intervienen en un lenguaje de programación, las palabras reservadas del mismo lenguaje, las variables y constantes que incluya el código fuente a transformar.

En este trabajo se proponen dos transformaciones, la primera: *codificación abstracta*, tiene como propósito identificar conjuntamente, los operadores que intervienen dentro de una categoría, de la clasificación de los operadores del lenguaje de programación seleccionado. La segunda transformación: *codificación en detalle*, tiene como fin, identificar de forma única, cada operador de la misma clasificación.

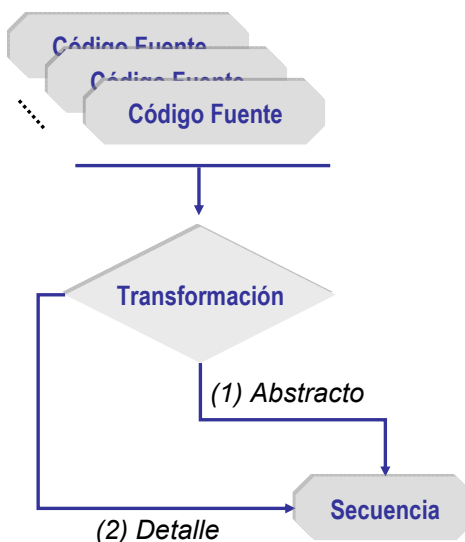


Figura 3.6. Cambio de representación propuesto.

## Codificación Abstracta

En este trabajo, se utilizó el lenguaje *C#* de la plataforma *.NET* de *Microsoft*®. La documentación del *.NET Framework* muestra la clasificación de operadores que intervienen en este lenguaje. [MSDN05] En la Figura 3.7, se muestra un listado de los operadores que intervienen en el lenguaje de programación mencionado.

Operator category	Operators
Arithmetic	+ - * / %
Logical (Boolean and bitwise)	&   ^ ! ~ &&    true false
String concatenation	+
Increment, decrement	++ --
Shift	<< >>
Relational	== != < > <= >=
Assignment	= += -= *= /= %= &=  = ^= <<= >>= ??
Member access	.
Indexing	[ ]
Cast	()
Conditional	? :
Delegate concatenation and removal	+ -
Object creation	New
Type information	as is sizeof typeof
Overflow exception control	checked unchecked
Indirection and Address	* -> [] &

**Figura 3.7.** Clasificación de operadores propuestos por Microsoft. [MSDN05]

No es el propósito de este trabajo el realizar un análisis extenso sobre esta especificación, es por ello que se redujo el conjunto de operadores utilizados en los

experimentos, preservando los de uso frecuente. Sin embargo se preserva la clasificación propuesta por Microsoft. Ver Figura 3.8.

Categoría de Operadores	Operador
Aritméticos	+ - * / %
Lógicos	&   ^ ! ~ &&    true false
Incremento, Decremento	++ --
De Cambio	<< >>
Relacionales	== != < > <= >=
Asignación	= += -= *= /= %= &=  = ^= <<= >>=
Indexado	[ ]
Cast	()
Creación de Objetos	new
Información de Tipo	as is sizeof typeof
Control de errores	checked unchecked

**Figura 3.8.** Categorías de operadores utilizados.

Para la codificación abstracta, se utilizaron un conjunto de reglas mostradas a continuación.

1. Identificar los operadores que intervienen en el lenguaje de programación seleccionado.
2. Designar las categorías de operadores a utilizar.
3. Determinar el valor inicial  $\theta \in \mathbf{R}$ , a partir del cuál comenzará la codificación.
4. Determinar el valor de incremento  $\Delta \in \mathbf{R}$ .
5. Asignar el valor  $\theta_i$  a cada operador utilizando las siguientes convenciones.
  - 5.1. Al primer operador de la primera categoría de operadores  $\theta_1 = \theta$
  - 5.2. Los operadores que pertenezcan a la misma categoría asignarles un valor único definido como:  $\theta_i = \theta_{i-1} + \Delta$  con  $i > 1$ .
  - 5.3. El valor del primer operador de cada categoría se encuentra definido como:  $\theta_i = \theta_{i-1} + \theta$ , donde  $\theta_{i-1}$  corresponde al último operador de la categoría anterior.

6. El valor  $\theta_i$  del primer operador de cada categoría, será utilizado como identificador de la categoría  $\phi_j$ .

Siguiendo estas reglas, se obtienen valores equidistantes de identificación para cada operador dentro de una categoría; de igual forma, se obtienen valores representativos para cada categoría, tomando en cuenta el número de operadores que intervienen en la clasificación de operadores. En la Figura 3.9, se muestra parte de la clasificación utilizada con valores  $\theta = 1.00$  y  $\Delta = 0.05$ , estos valores fueron designados experimentalmente para controlar los valores de similitud arrojados por FDTW. En el Capítulo 5, se muestran resultados obtenidos con diferentes valores de  $\theta$  y  $\Delta$ .

<b>Aritméticos</b>	<b>Valor</b>	<b><math>\phi_1 = 1.00</math></b>
+	$\theta_1 = \theta$	1.00
-	$\theta_2 = \theta_1 + \Delta$	1.05
*	$\theta_3 = \theta_2 + \Delta$	1.10
/	$\theta_4 = \theta_3 + \Delta$	1.15
%	$\theta_5 = \theta_4 + \Delta$	1.20
<b>Lógicos</b>	<b>Valor</b>	<b><math>\phi_2 = 2.20</math></b>
&	$\theta_6 = \theta_5 + \theta$	2.20
	$\theta_7 = \theta_6 + \Delta$	2.25
^	$\theta_8 = \theta_7 + \Delta$	2.30
!	$\theta_9 = \theta_8 + \Delta$	2.35
~	$\theta_{10} = \theta_9 + \Delta$	2.40
&&	$\theta_{11} = \theta_{10} + \Delta$	2.45
	$\theta_{12} = \theta_{11} + \Delta$	2.50
true	$\theta_{13} = \theta_{12} + \Delta$	2.55
false	$\theta_{14} = \theta_{13} + \Delta$	2.60
<b>Incremento / Decremento</b>	<b>Valor</b>	<b><math>\phi_3 = 3.60</math></b>
++	$\theta_{15} = \theta_{14} + \theta$	3.60
--	$\theta_{16} = \theta_{15} + \Delta$	3.65

**Figura 3.9.** Valores asignados a los operadores con  $\theta = 1.00$  y  $\Delta = 0.05$

Una vez designados los valores para cada operador  $\theta_i$ , y para cada categoría  $\phi_j$ ; se sustituyen los operadores involucrados en cada instrucción, del código fuente a transformar, por el valor  $\phi_j$  de la categoría a la que pertenezca.

En la Figura 3.10a se muestra un código en C# que tiene como función el amplificar una imagen representada por un objeto de la Clase Bitmap, dado un factor de escala.

- Nivel 1**
- Llamada Función
  - Asignación
  - Asignación
  - Creación de Objeto
  - Llamada Función
  - Op\_Aritmetica
  - Op\_Aritmetica
  - Asignación
  - Op\_Relacional
  - Incremento
  - Asignación
  - Op\_Relacional
  - Incremento
  - Asignación
  - Llamada función
  - Asignación
  - Op\_Relacional
  - Incremento
  - Asignación
  - Op\_Relacional
  - Incremento
  - Llamada Funcion
  - Op\_Aritmetica
  - Op\_Aritmetica
  - Op\_Aritmetica
  - Op\_Aritmetica
  - Llamada Funcion
  - Op\_Aritmetica
  - Op\_Aritmetica

```
private void Amplificar(NewImage active,int factor)
{
    Bitmap old = active.input;
    Bitmap NewImg = new Bitmap(old.Width * factor,old.Height * factor);

    for (int i = 0; i < old.Size.Width; i++)
    {
        for (int j = 0; j < old.Size.Height; j++)
        {
            Color c = old.GetPixel(i, j);
            for (int x = 0; x < factor; x++)
            {
                for (int y = 0; y < factor; y++)
                {
                    NewImg.SetPixel(i * factor + x, j * factor + y, c);
                }
            }
        }
        SetProgress((i / old.Width) * 100);
    }
}
```

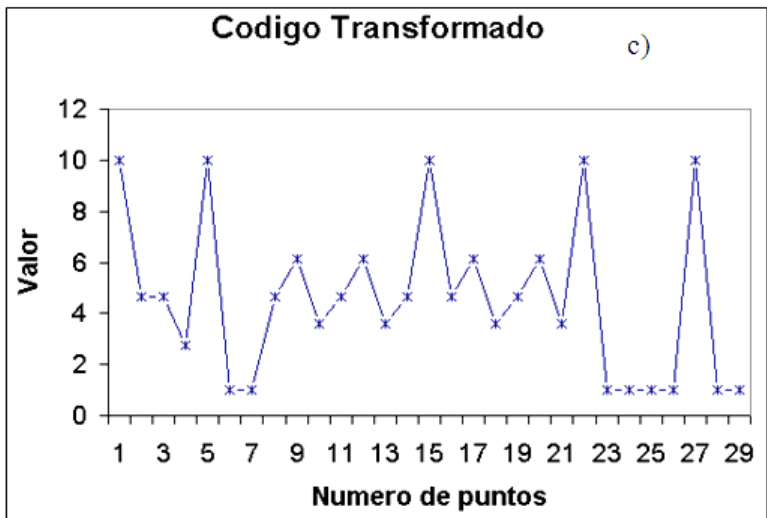


Figura 3.10. Código fuente de entrada (a), representación en primer nivel (b), secuencia obtenida utilizando codificación abstracta (c).

## Codificación en Detalle

Para transformar el código utilizando la codificación en detalle, se sustituyen los operadores que intervienen en cada instrucción del código fuente, por los valores  $\theta_i$  designados con las reglas mostradas en la sección anterior. Además, cada palabra reservada del lenguaje de programación se le asigna un valor  $\sigma \in \mathbf{R}$ . Los valores  $\sigma$ , son *equi-distantes* entre el conjunto de palabras reservadas.

Para realizar los experimentos, las palabras reservadas fueron clasificadas dentro de las categorías denominadas: *Tipos de datos*, *Ciclos*, *Modificadores de Acceso*, *Declarativas*, *Cast*, *Comparaciones*, *Excepciones*, *Sentencias*, *No seguras*, *Contextuales*, *Otras*. Al ordenar las palabras reservadas, en forma consecutiva de acuerdo a esta clasificación, les fueron asignados valores *equi-distantes*, diferentes de los valores que corresponden a los operadores y las categorías de operadores. Ver Figura 3.11.

Clase	Palabras Reservadas
<i>Tipos de Datos</i>	void byte sbyte int uint short ushort long ulong float double decimal bool char string
<i>Ciclos</i>	for foreach in do while break continue
<i>Modificadores de Acceso</i>	public private protected static virtual abstract base const delegate volatile readonly value override implicit extern event explicit internal
<i>Declarativas</i>	namespace class struct new enum interface lock sealed operator using object this
<i>Cast</i>	as is
<i>Comparaciones</i>	if else switch case default return goto



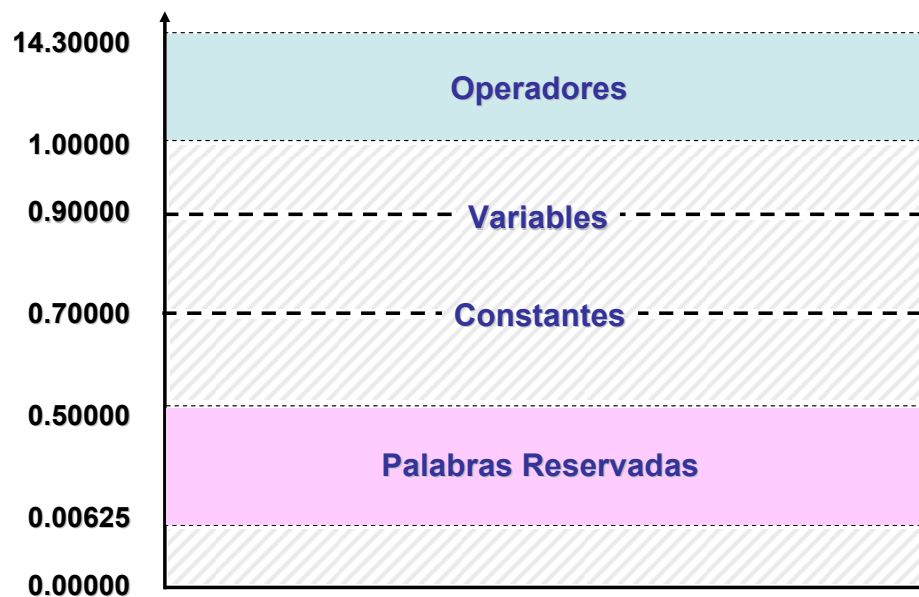
<i>Excepciones</i>	try catch finally throw
<i>Sentencias</i>	checked unchecked
<i>No Seguras</i>	fixed unsafe
<i>Contextuales</i>	get value partial where set yield
<i>Otras</i>	out ref params sizeof stackalloc typeof

**Figura 3.11.** Palabras reservadas utilizadas para realizar la transformación a nivel de operadores.

Las reglas para asignar los valores a las palabras reservadas, son las siguientes:

1. Seleccionar un valor  $\delta$  inicial.
2. La primer palabra reservada  $\sigma_1 = \delta$ .
3. Para cada palabra reservada  $\sigma_i = \sigma_{i-1} + \delta$ , con  $i > 1$ .

En los experimentos se utilizó  $\delta = 0.00625$ , obtenido de la división entre el número de palabras reservadas y el intervalo  $(0,5]$ . Esto es, para obtener un conjunto de valores diferentes a los valores de los operadores mostrados en la sección anterior de este capítulo. Ver Figura 3.12.



**Figura 3.12.** Representación gráfica de los valores utilizados

Tipos de datos		Valor
void	$\sigma_1 = \delta$	0.00625
byte	$\sigma_2 = \sigma_1 + \delta$	0.01250
sbyte	$\sigma_3 = \sigma_2 + \delta$	0.01875
Int	$\sigma_4 = \sigma_3 + \delta$	0.02500
uint	$\sigma_5 = \sigma_4 + \delta$	0.03125
short	$\sigma_6 = \sigma_5 + \delta$	0.03750
ushort	$\sigma_7 = \sigma_6 + \delta$	0.04375
long	$\sigma_8 = \sigma_7 + \delta$	0.05000
ulong	$\sigma_9 = \sigma_8 + \delta$	0.05625
float	$\sigma_{10} = \sigma_9 + \delta$	0.06250
double	$\sigma_{11} = \sigma_{10} + \delta$	0.06875
decimal	$\sigma_{12} = \sigma_{11} + \delta$	0.07500
bool	$\sigma_{13} = \sigma_{12} + \delta$	0.08125
char	$\sigma_{14} = \sigma_{13} + \delta$	0.08750
string	$\sigma_{15} = \sigma_{14} + \delta$	0.09375

Ciclos		Valor
for	$\sigma_{16} = \sigma_{15} + \delta$	0.10000
foreach	$\sigma_{17} = \sigma_{16} + \delta$	0.10625
in	$\sigma_{18} = \sigma_{17} + \delta$	0.11250
do	$\sigma_{19} = \sigma_{18} + \delta$	0.11875
while	$\sigma_{20} = \sigma_{19} + \delta$	0.12500
break	$\sigma_{21} = \sigma_{20} + \delta$	0.13125
continue	$\sigma_{22} = \sigma_{21} + \delta$	0.13750

Modificadores de Acceso		Valor
public	$\sigma_{18} = \sigma_{17} + \delta$	0.14375
private	$\sigma_{19} = \sigma_{18} + \delta$	0.15000
protected	$\sigma_{20} = \sigma_{19} + \delta$	0.15625
static	$\sigma_{21} = \sigma_{20} + \delta$	0.16250
virtual	$\sigma_{22} = \sigma_{21} + \delta$	0.16875
...	...	...

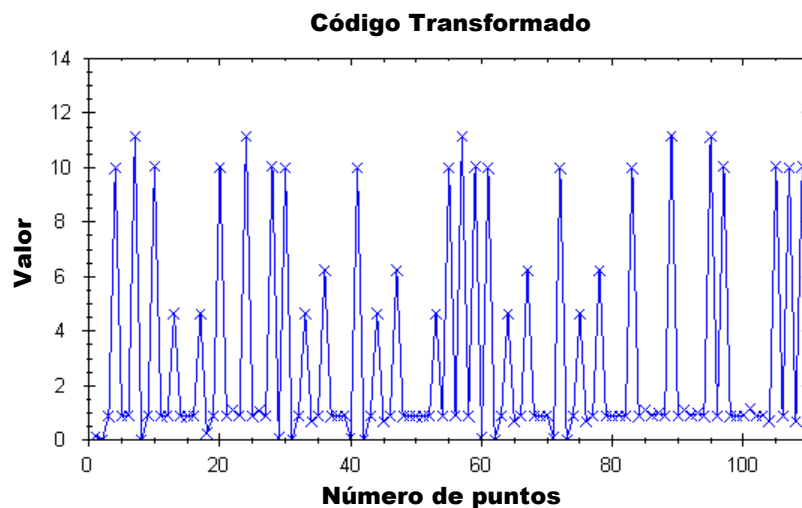
**Figura 3.13.** Palabras reservadas utilizadas para realizar la codificación en detalle, con  $\sigma = 0.00625$ .

Teniendo en cuenta las especificaciones anteriormente mencionadas, la intersección de los valores  $\theta \cap \sigma = \{\}^3$ . Esto es para evitar ambigüedad entre los operadores, y las palabras reservadas. Para la codificación en detalle, se sustituyen las variables con un valor de 0.7 y las constantes por un valor de 0.9. Estos valores fueron designados experimentalmente, sin embargo pueden sustituirse por otros valores, mientras sean diferentes de los valores de los operadores y de los valores asignados a las palabras reservadas.

<sup>3</sup> Denota el conjunto vacío.

Con la representación del código utilizando la **codificación abstracta**, se obtiene la estructura esencial del programa a nivel de instrucciones. Con la **codificación en detalle** se obtiene una estructura del código más detallada.

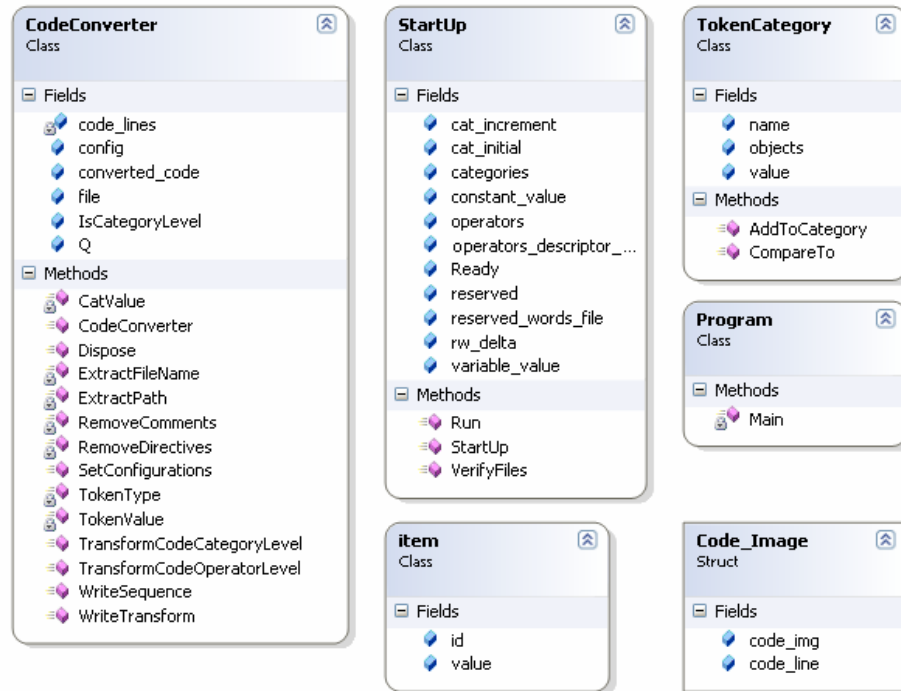
En la Figura 3.14. se muestra la representación del código fuente de la Figura 3.10 utilizando la codificación en detalle. En esta representación se incluyen las palabras reservadas, los operadores del lenguaje de programación, las variables y constantes utilizadas en el código de entrada.



**Figura 3.14.** Código fuente de la Figura 3.10. codificado en detalle.

Los algoritmos que realizan estas conversiones se muestran en el Anexo. El método que realiza la transformación utilizando la codificación abstracta [a nivel de categoría de operadores] es *TransformCodeCategoryLevel*.

El método *TransformCodeOperatorLevel*, realiza la transformación utilizando la codificación en detalle [a nivel operadores]. Ambos métodos pertenecen a la clase *CodeConverter*. En la Figura 3.15 se muestra una estructura conceptual de la clase mencionada anteriormente.



**Figura 3.15.** Diagrama de clases del proyecto *CSharpCodeConverter*.

La clase *StartUp* contiene los operadores, las categorías de operadores y las palabras reservadas definidas para aceptar un lenguaje de programación. El método *VerifyFiles*, comprueba la integridad de los archivos de entrada. El método *Run*, realiza la lectura de las categorías de operadores y utiliza la estructura *TokenCategory*, para almacenar los operadores. Las clases *item* y *Code\_Image* son utilizadas para almacenar los valores individuales de un operador y la correspondencia en código fuente del valor de la transformación respectivamente.

Siguiendo las reglas mencionadas en las secciones previas de este trabajo, es relativamente sencillo implementar un lenguaje de programación diferente a C# para ser reconocido por el programa de conversión.

La clase *CodeConverter*, es el centro de la transformación, los métodos *RemoveComments* y *RemoveDirectives*, eliminan los comentarios y directivas de inclusión del código fuente de entrada. Los métodos: *TransformCodeCategoryLevel*, y *TransformCodeOperatorLevel*, deben ser “invocados” de forma exclusiva. Los

métodos incluidos en este proyecto se encuentran debidamente documentados en el CD correspondiente a esta tesis.

## Referencias del capítulo

[Silverman&Morgan90] Silverman, H. F. & Morgan, “*The application of dynamic programming to connected speech recognition*”. IEEE ASSP Magazine, 7--24. D. P. July, 1990.

[Keogh02] Keogh E. “*Exact Indexing of Dynamic Time Warping*”. In Proceedings of the 28<sup>th</sup> VLDB Conference, pp. 406-417, 2002.

[Berndt96] Berndt D. J., Clifford J. “*Finding Patterns in Time Series: A Dynamic Programming Approach*”. Advances in Knowledge Discovery and Data Mining, AAAI/MIT, pp. 229-248, 1996.

[Angeles05e] Angeles-Yreta A., Figueroa-Nazuno J., Ramírez-Amaro K. “*Búsqueda de Semejanza entre Objetos 3D por Indexado*”. Decimosexta Reunión de Otoño Comunicaciones, Computación, Electrónica y Exposición Industrial, ROC&C’ 2005.

[Mirón-Bernal06] M. Mirón-Bernal, A. Angeles-Yreta, J. Figueroa-Nazuno, “*Clasificación de Series de Tiempo mediante la identificación de subsecuencias similares*”, Memorias del XLIXI Congreso Nacional de Física. S.L.P, S.L.P. ISSN 0187-4713. Octubre 2006,

[Jang00] Jang J.S.R, Gao M.Y. “*A Query-by-Singing System based on Dynamic Programming*”. International Workshop on Systems Resolutions (the 8<sup>th</sup> Bellman Continuum), pp. 85-89, 2000.

[Kim01] Kim S. W., Park S., Chu W. W. “*An Index-Based Approach for Similarity Search Supporting Time Warping in Large Sequence Database*”. In Proceedings of the 17<sup>th</sup> International Conference on Data Engineering (ICDE’01), pp. 607-614, 2001.

[Kamejima97] Kohji Kamejima, “*Parallel Distributed Scheme for Stochastic Coding of Self-similar Patterns,*” pas, p. 192, 2nd AIZU International Symposium on Parallel Algorithms / Architecture Synthesis (pAs '97), 1997

[MSDN05] The Visual Studio Combined Help Collection. Microsoft Visual Studio 2005 Documentation. Microsoft Corporation.

## **CAPÍTULO 4**

---

# **EL CONJUNTO DE DATOS**

# Creación Del Conjunto De Datos

## Usando Códigos Fuente

El conjunto de datos esta compuesto por 210 códigos en C#. En las Tablas 4.1 , 4.2 y 4.3, se muestran las diferentes funciones que realizan los códigos fuente. Los códigos recopilados son asignaciones de clase del Instituto Politécnico Nacional. Para conformar el conjunto de datos, se realizaron modificaciones sistemáticas a los códigos fuente como: reemplazo de tipos de datos, modificación en el orden de instrucciones, entre otras variaciones, para obtener códigos *similares* de referencia a cada código fuente original. Los códigos de entrada se encuentran libres de errores de sintaxis. El conjunto de datos se encuentra integrado dentro del CD correspondiente a esta tesis.

<b>Clases</b>	<b>Propósito</b>	<b>Cantidad</b>
AMP	<i>Amplificar el tamaño de una imagen</i>	12
BW	<i>Convertir una imagen en escala de grises a blanco y negro</i>	4
DMA	<i>Obtener el mapa de distancias de una imagen binaria</i>	4
ED	<i>Detección de bordes de una imagen binaria</i>	4
GS	<i>Conversión de una imagen en colores a escala de grises</i>	2
HG	<i>Obtener el Histograma de una imagen</i>	3
RAW	<i>Convertir un archivo binario a imagen</i>	2
RD	<i>Reducir el tamaño de una imagen</i>	2

**Tabla 4.1.** Códigos fuente obtenidos de clases de *Procesamiento Digital de Imágenes* del *Centro de Investigación del IPN*.



Clases	Propósito	Cantidad
Cliente	Envío de mensajes usando Sockets	3
Servidor	Envío/ Recepción de mensajes usando Sockets	3

**Tabla 4.2.** Códigos fuente obtenidos de clases de *Sistemas Operativos* del Centro de Investigación del IPN.

Clases	Propósito	Cantidad
Rumelhart	Algoritmo Rumelhart and McClelland	2
NN	Algoritmo Red Neuronal	1

**Tabla 4.3.** Códigos fuente obtenidos de clases de *Inteligencia Artificial* del Centro de Investigación del IPN.

A continuación se muestra parte del conjunto de datos.

```
private void Amplificar(NewImage active,int factor)
{
    Bitmap old = active.input;
    Bitmap NewImg = new Bitmap(old.Width * factor,old.Height * factor);

    for (int i = 0; i < old.Size.Width; i++)
    {
        for (int j = 0; j < old.Size.Height; j++)
        {
            Color c = old.GetPixel(i, j);
            for (int x = 0; x < factor; x++)
            {
                for (int y = 0; y < factor; y++)
                    NewImg.SetPixel(i * factor + x, j * factor + y, c);
            }
        }
        SetProgress((i / old.Width) * 100);
    }
}
```

**Figura 4.1.** Código en C# - AMP01, su función es amplificar una imagen dentro de un objeto de la clase Bitmap, dado un factor de escala.

```

public void Resize(int factor)
{
    Bitmap input = this.input;
    int i = 0; int j = 0;
    Bitmap output = new Bitmap(input.Width * factor, input.Height * factor);

    for (i = 0; i < input.Size.Width; i++)
    {
        for (j = 0; j < input.Size.Height; j++)
        {
            Color pixel = input.GetPixel(i, j);
            for (int x = 0; x < factor; x++)
            {
                for (int y = 0; y < factor; y++)
                    output.SetPixel(i * factor + x, j * factor + y, pixel);
            }
        }
    }
}

```

**Figura 4.2.** Código en C# - AMP11, este código muestra la onceava variación sobre el código fuente AMP01.

```

private void OpenRawImage()
{
    FileStream fs = new FileStream(file, FileMode.Open, FileAccess.Read);
    BinaryReader r = new BinaryReader(fs);
    byte[] temp = new byte[fs.Length];
    for (int i = 0; i < fs.Length; i++)
        temp[i] = r.ReadByte();

    r.Close();
    fs.Close();

    int RawWidth, RawHeight;
    RawWidth = Convert.ToInt32(txtWidth.Text);
    RawHeight = Convert.ToInt32(txtHeight.Text);
    Bitmap input = new Bitmap(RawWidth, RawHeight, PixelFormat.Format8bppIndexed);
    unsafe
    {
        BitmapData bmData = input.LockBits(
            new Rectangle(0, 0, input.Width, input.Height),
            ImageLockMode.WriteOnly,
            PixelFormat.Format8bppIndexed);
        byte* p = (byte*)bmData.Scan0;
        int diff = bmData.Stride - input.Width;

        for (int i = 0; i < temp.Length; i++)
        {
            *p++ = temp[i];
            if (i % input.Width == 0 && i > 0)
                p += diff;
        }
        input.UnlockBits(bmData);
    }
}

```

**Figura 4.3.** Código en C# - RAW01, este código tiene como función abrir una imagen en escala de gris codificada en un archivo binario.

Los códigos fuente del conjunto de datos, se encuentran incluidos dentro del CD correspondiente a este trabajo.

## Referencias del capítulo

[Mirón-Bernal05] M. Mirón-Bernal, A. Angeles-Yreta, J. Figueroa-Nazuno, “*Complejidad y Dependencias de Rango Extendido en Series de Tiempo*”, Memorias del XLVIII Congreso Nacional de Física. Guadalajara, Guadalajara. ISSN 0187-4713. Octubre 2005.

[Karagiannis04]T. Karagiannis, M.Molle, M.Faloutsos, “*Long Range Dependence, ten years of Internet Traffic Modeling*”, IEEE Internet Computing, September-October 2004.

[Ramírez04]K. Ramírez-Amaro, H. Jiménez-Hernández, J. Figueroa-Nazuno, “*Análisis de los Índices de Teleconexión Atmosféricos Empleando Técnicas de Mapas Recurrentes Y Modelos Casuales*”; IEEE Reunión de Otoño de Comunicaciones y Computación, Noviembre 2004.

[Keogh02] Keogh E. “*Exact Indexing of Dynamic Time Warping*”. In Proceedings of the 28<sup>th</sup> VLDB Conference, pp. 406-417, 2002.

[Matsumoto98] M. Matsumoto & T. Nishimura, “*Mersenne twister: a 623-dimensionally equidistributed uniform pseudorandom number generator*”, ACM Trans. Model. Comput. Simul. 8, 3 (1998).

# **CAPÍTULO 5**

---

## **RESULTADOS**

## Comparación de Códigos Fuente

En esta Sección, se muestran los resultados obtenidos al comparar el conjunto de datos de los códigos fuente mencionados en el Capítulo 4, con *Fast Dynamic Time Warping (FDTW)*. Para realizar esta comparación, se utilizaron las transformaciones propuestas en el Capítulo 3 de este trabajo.

En el primer apartado de esta sección, se muestran los resultados obtenidos transformando el código fuente utilizando la codificación abstracta; en el segundo apartado, se propone un valor de similitud entre códigos fuente; en el tercer apartado, se muestran los resultados obtenidos de la comparación de códigos fuente utilizando la codificación en detalle; en el cuarto apartado, se muestran los resultados obtenidos con diferentes valores de los parámetros de codificación; finalmente, se muestran los resultados obtenidos con diferentes clasificaciones de operadores.

### Codificación Abstracta

En este apartado, se muestran los resultados utilizando los valores  $\theta = 1.00$  y  $\Delta = 0.05$ , que corresponden al inicio de la codificación y el incremento para cada operador que interviene en la clasificación de operadores del lenguaje de programación. La explicación de los valores, que intervienen en la codificación abstracta, se encuentra en el Capítulo 3.

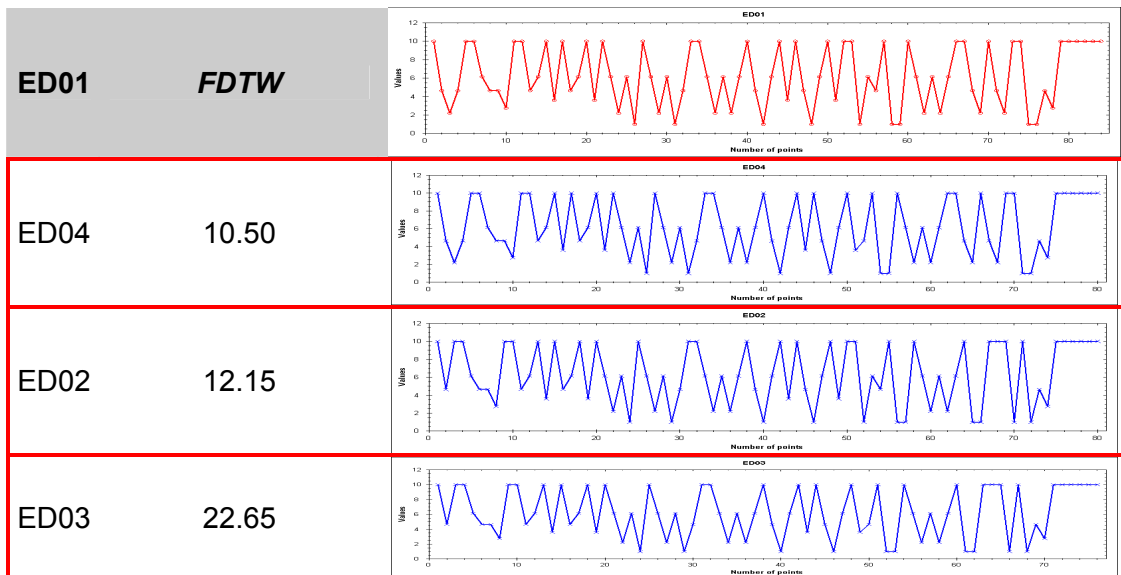
En la Tabla 5.1, se muestra resaltado en negritas un código de entrada  $Q$ , (*Query*) y resaltados con borde rojo, los tres programas más similares a  $Q$ . En el primer ejemplo, el código de entrada ED01, es parecido a ED04 con  $FDTW=10.50$ , a ED02 con  $FDTW=12.15$ , y ED03 con  $FDTW=22.65$ . Como se explico en el Capítulo 4, estos códigos fuente pertenecen a una clase que detecta los bordes de una imagen binaria. De igual forma, se muestran los códigos similares a *DMA04*, *Server02* y *AMP04*.

	<b>ED01</b>		<b>DMA04</b>		<b>Server02</b>		<b>AMP04</b>
ED01	0.00	DMA04	0.00	Server02	0.00	AMP04	0.00
ED04	10.50	DMA01	66.95	Server03	23.60	AMP09	12.80
ED02	12.15	DMA02	78.30	Server01	30.65	AMP03	15.65
ED03	22.65	DMA03	107.50	Cliente03	32.70	AMP05	15.80
RD02	75.75	Rumelhart02	254.70	BW02	39.85	AMP06	15.80
RAW01	96.70	NN01	257.75	BW04	39.85	AMP08	28.60
RAW02	96.70	Rumelhart01	270.05	HG01	42.35	AMP11	34.05
RD01	114.75	ED01	307.95	GS01	43.10	AMP12	34.05
Cliente02	116.40	ED04	316.15	Cliente02	43.95	AMP02	39.40
BW01	119.30	ED02	327.80	BW01	50.25	AMP07	39.40
...	...	...	...	...	...	...	...

**Tabla 5.1.** Comparación de códigos fuente dado un *Query* en el conjunto de datos. Se muestran resaltados con borde rojo, los tres códigos más similares utilizando  $FDTW$ .

En la Figura 5.1, se muestran las secuencias numéricas representativas de los códigos *ED01*, *ED02*, *ED03*, y *ED04*. Estos códigos realizan la detección de bordes en una imagen binaria. La secuencia en color rojo muestra el código de referencia *Query*. Las secuencias en color azul, son la representación gráfica de los tres códigos más similares al código *ED01*.

El código *ED01*, realiza la extracción de bordes de una imagen binaria, utilizando los cuatro vecinos de cada píxel de la imagen. El código *ED04*, también realiza la extracción de bordes de la imagen, a diferencia de *ED01*, utiliza los ocho vecinos de cada píxel de la imagen. Ambos códigos se muestran en las Figuras 5.5 y 5.6, respectivamente.



**Figura 5.1.** Representación gráfica de los tres códigos similares al código de referencia *ED01*.

En la Figura 5.2, se muestran las Secuencias numéricas representativas de los códigos *AMP04*, *AMP09*, *AMP03*, y *AMP05*. La secuencia en color rojo muestra el código de referencia *Query*. Las secuencias en color azul, son la representación gráfica de los tres códigos más similares al código *AMP04*. Estos códigos, pertenecen al conjunto de datos compuestos por programas realizados en clases de *Procesamiento de Imágenes*, descritos en el Capítulo 4 de este trabajo. La clase *AMP*, tiene como propósito ampliar una imagen dado un factor de escala.

En la Figura 5.3, se muestran los tres códigos similares al código de referencia *Server02*, el código fuente con mayor similitud es *Server03* con distancia *FDTW*=23.60. El segundo código similar es *Server01* con *FDTW*= 30.65; finalmente el tercer código similar es *Ciente03* con *FDTW*= 32.70.

Los códigos *Server02* y *Server03* son mostrados en las Figuras 5.26 y 5.27 respectivamente.

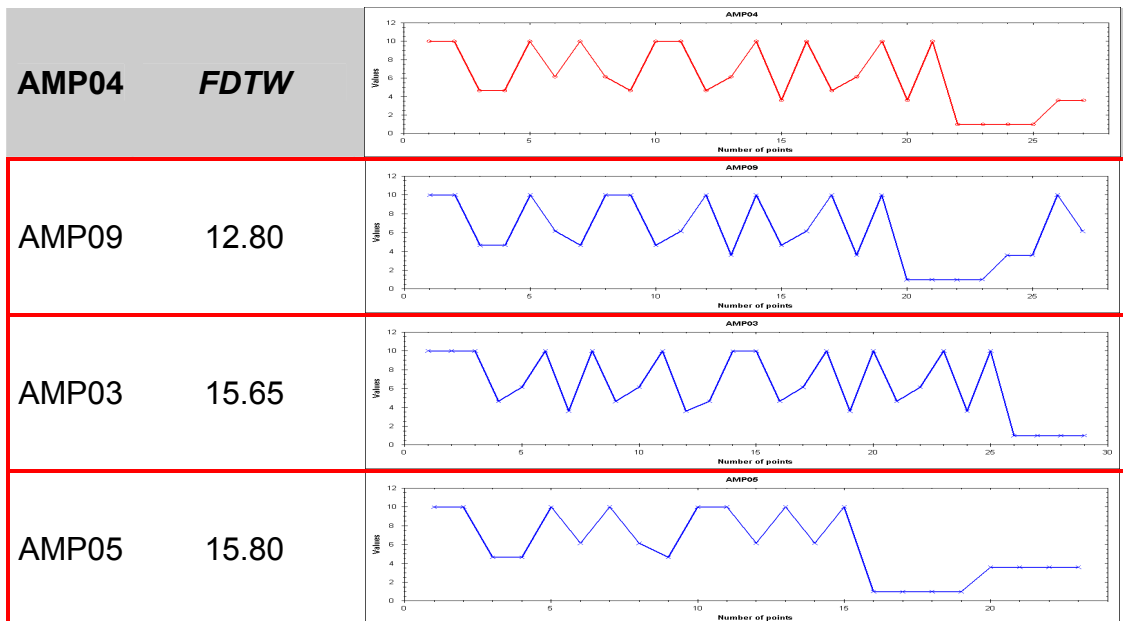


Figura 5.2. Representación gráfica de los tres códigos similares al código de referencia *AMP04*.

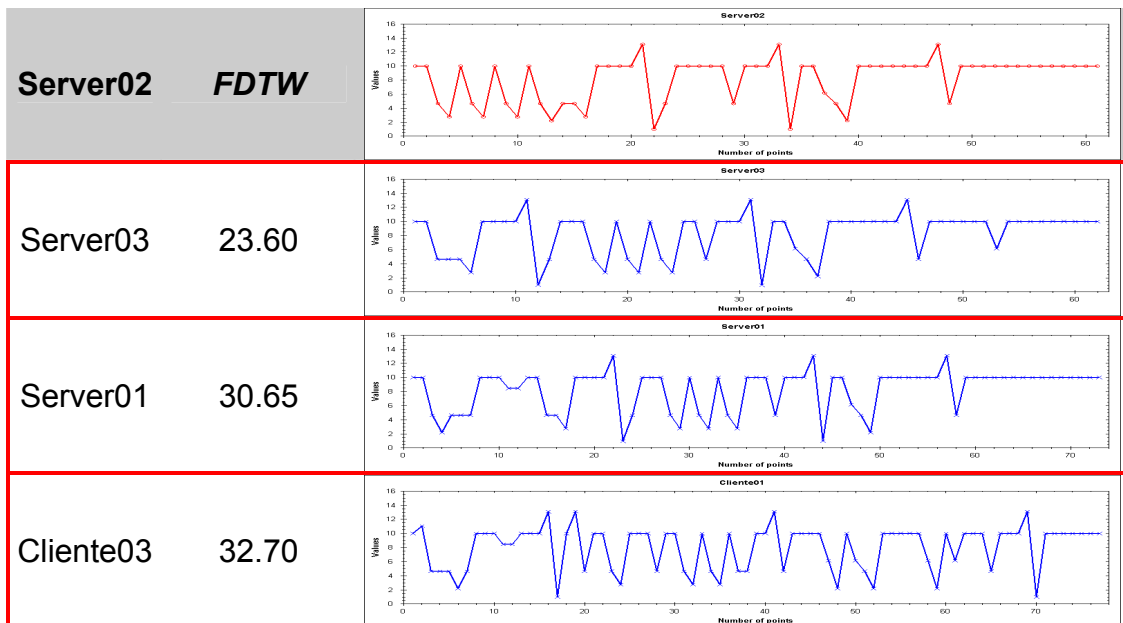
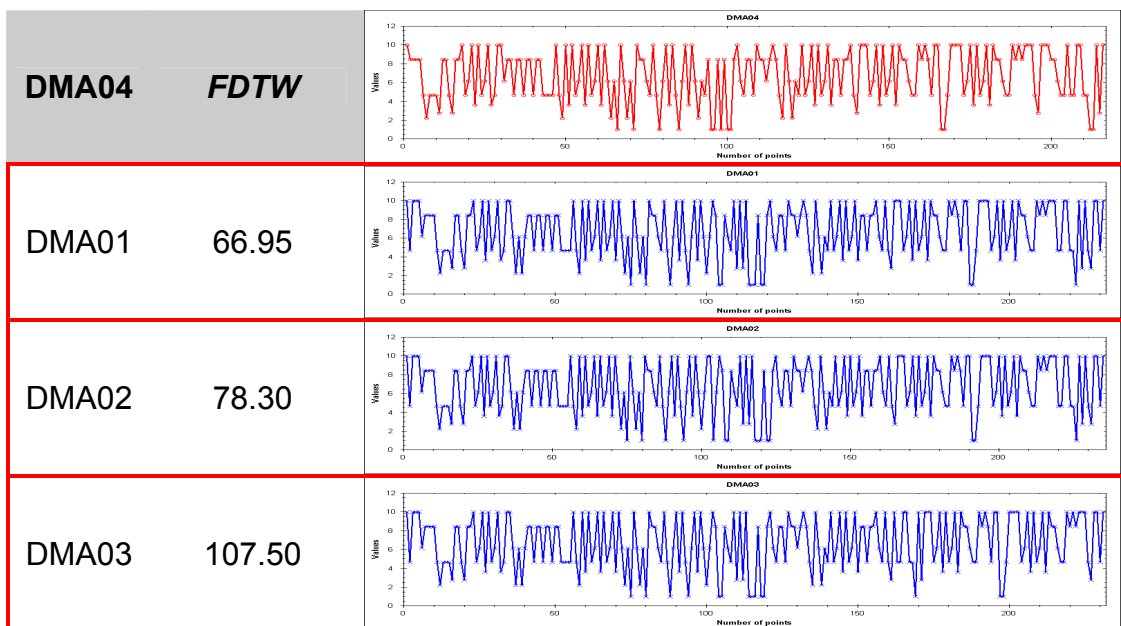


Figura 5.3. Representación gráfica de los tres códigos similares al código de referencia *Server02*.



Como se observa en la Figura 5.3, el tercer código similar a *Server02*, pertenece a un código con propósito diferente a la clase *Server*, descrita en el Capítulo 4. El código *Server02*, realiza la comunicación mediante *sockets* a un código de la clase *Cliente*. Los códigos fuente *Cliente*, se comunican a códigos de la clase *Server* mediante *sockets*. Aún cuando realizan funciones diferentes, la estructura es similar ya que ambos utilizan un medio de comunicación común entre ellos para enviar o recibir datos. El código *Cliente03*, se muestra en la Figura 5.25.



**Figura 5.4.** Representación gráfica de los tres códigos similares al código de referencia *DMA04*.

En la Figura 5.4, se muestran las representaciones en secuencia numérica de los tres códigos similares a *DMA04*, dentro del conjunto de datos. Los códigos que pertenecen a la Clase *DMA*, tienen como propósito obtener el mapa de distancias de una imagen binaria, estos programas utilizan como distancia base: la distancia *euclidiana*, la distancia *manhatan*, y la distancia *chessboard*.

Los códigos mostrados en este trabajo, al igual que el conjunto de datos descritos en el Capítulo 4, se encuentran incluidos dentro del CD correspondiente a este trabajo.

```

private void vecindadToolStripMenuItem_Click(object sender, EventArgs e)
{
    //Deteccion de bordes en imagenes binarias
    bool found = false;
    try
    {
        NewImage temp = (NewImage)ActiveMdiChild;
        if (temp != null)
        {
            Bitmap input, output;
            input = temp.input;
            output = new Bitmap(input.Width, input.Height);
            for (int i = 0; i < input.Size.Width; i++)
            {
                for (int j = 0; j < input.Size.Height; j++)
                {
                    // Limites de la imagen
                    if (j == 0 || j == input.Size.Height - 1)
                        ;
                    else if (i == 0 || i == input.Size.Width - 1)
                        break;
                    Color p = input.GetPixel(i, j);
                    if (p.R == 0 && p.G == 0 && p.B == 0) //Es Fondo
                        continue;
                    else
                        //Buscar en los vecinos
                        for (int x = -1; x <= 1; x++)
                        {
                            for (int y = -1; y <= 1; y++)
                            {
                                if ((x * y) != 0)
                                    continue;
                                Color q = input.GetPixel(x + i, y + j);
                                //Si un vecino Es Fondo
                                if (q.R == 0 && q.G == 0 && q.B == 0)
                                {
                                    //Entonces es un borde interno
                                    output.SetPixel(i, j, Color.FromArgb(
                                        p.R, p.G, p.B));
                                    // Para agilizar interno
                                    found = true;
                                    break;
                                }
                                //Entonces es un borde externo
                                //output.SetPixel(
                                //x+i, y+j, Color.FromArgb(
                                //q.R, q.G, q.B));
                            }
                        }
                    //Interno para agilizar
                    if (found)
                    {
                        found = false;
                        break;
                    }
                }
            }
            SetProgress((i / input.Width) * 100);
        }
        NewImage salida = new NewImage(output);
        salida.SetParent(this);
        salida.ChangeName("BDI4", temp.name);
        SetProgress(100);
    }
    (Exception ex)
    {
        SetProgress(100);MessageBox.Show(ex.ToString());
    }
}

```

**Figura 5.5.** Programa *ED01*. Detecta bordes en una imagen binaria utilizando cuatro vecinos.

```

private void vecindadToolStripMenuItem1_Click(object sender, EventArgs e)
{
    bool found = false;
    try
    {
        NewImage temp = (NewImage)ActiveMdiChild;
        if (temp != null)
        {
            Bitmap input, output;
            input = temp.input;
            output = new Bitmap(input.Width, input.Height);
            for (int i = 0; i < input.Size.Width; i++)
            {
                for (int j = 0; j < input.Size.Height; j++)
                {
                    // Límites de la imagen
                    if (j == 0 || j == input.Size.Height - 1)
                        continue;
                    else if (i == 0 || i == input.Size.Width - 1)
                        break;

                    Color p = input.GetPixel(i, j);
                    if (p.R == 0 && p.G == 0 && p.B == 0) //Es Fondo
                        continue;
                    else
                        //Buscar en los vecinos
                        for (int x = -1; x <= 1; x++)
                        {
                            for (int y = -1; y <= 1; y++)
                            {
                                Color q = input.GetPixel(x + i, y + j);

                                if (q.R == 0 && q.G == 0 && q.B == 0)
                                    //Si un vecino Es Fondo
                                    {
                                        //Entonces es un borde interno
                                        output.SetPixel(
                                            i, j, Color.FromArgb(p.R, p.G, p.B)
                                        );
                                        // Para agilizar interno
                                        found = true;
                                        break;
                                    }
                            }
                        }
                    if (found)
                    {
                        found = false;
                        break;
                    }
                }
            }
            SetProgress((i / input.Width) * 100);
        }
        NewImage salida = new NewImage(output);
        salida.SetParent(this);
        salida.ChangeName("BDIS", temp.name);
        SetProgress(100);
    }
    catch (Exception ex)
    {
        SetProgress(100);
    }
}

```

**Figura 5.6.** Programa *ED04*. Detecta bordes en una imagen binaria utilizando ocho vecinos.

Cliente01		HG02		Rumelhart02		BW01	
Cliente01	0.00	HG02	0.00	Rumelhart02	0.00	BW01	0.00
Cliente02	38.10	HG03	6.35	Rumelhart01	24.35	GS01	25.65
Cliente03	59.85	BW04	14.75	ED01	176.40	GS02	30.10
Server03	62.50	BW02	14.75	ED04	176.60	AMP01	32.40
Server01	63.00	BW03	17.10	ED02	182.15	HG01	37.00
AMP01	64.35	HG01	26.35	ED03	182.35	AMP09	39.00
BW01	72.90	AMP08	27.35	RAW02	186.20	BW02	39.05
Server02	76.55	AMP09	31.05	RAW01	186.20	BW04	39.05
GS01	85.60	AMP06	38.70	RD02	193.15	Cliente03	42.40
GS02	85.65	AMP05	38.70	NN01	210.35	Cliente02	45.50
...	...	...	...	...	...	...	...

**Tabla 5.2.** Comparación de códigos fuente dado un *Query* en el conjunto de datos. Se muestran resaltados con borde rojo, los tres códigos más similares utilizando *FDTW*.

En la Tabla 5.2, se presenta la continuación de resultados de la Tabla 5.1, donde se muestran los códigos similares a los códigos *Cliente01*, *HG02*, *Rumelhart02* y *BW01*, remarcados con borde rojo.

En la Figura 5.7, se muestran las representaciones en secuencias de los tres códigos similares al código de referencia *Cliente01*, el código fuente con mayor similitud es *Cliente02* con distancia *FDTW*=38.10. El segundo código similar es *Cliente03* con *FDTW*= 59.85; finalmente el tercer código similar es *Server03* con *FDTW*= 62.50. En este experimento, la razón de la posición del código *Server03*, en el tercer lugar es que no existen más códigos de la clase *Cliente* dentro del conjunto de datos obtenido. Los códigos *Cliente01* y *Cliente02* son mostrados en las Figuras 5.14 y 5.15 respectivamente.

En la Figura 5.8, se encuentran las secuencias numéricas correspondientes a los códigos similares a *HG02*. En la Tabla 5.2, puede observarse al código *HG01*, fuera de los tres primeros lugares con respecto a *HG02*, en la Figura 5.9, se muestra la representación del código *HG01*. El código fuente se encuentra incluido en del CD correspondiente a esta tesis.

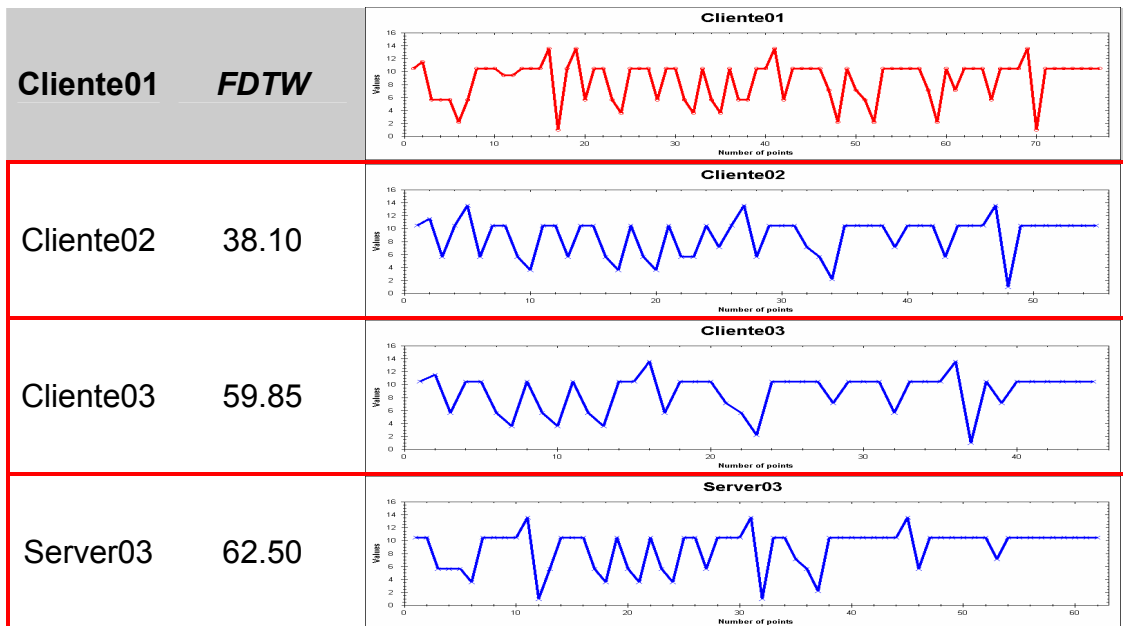


Figura 5.7. Representación gráfica de los tres códigos similares al código de referencia *Cliente01*.

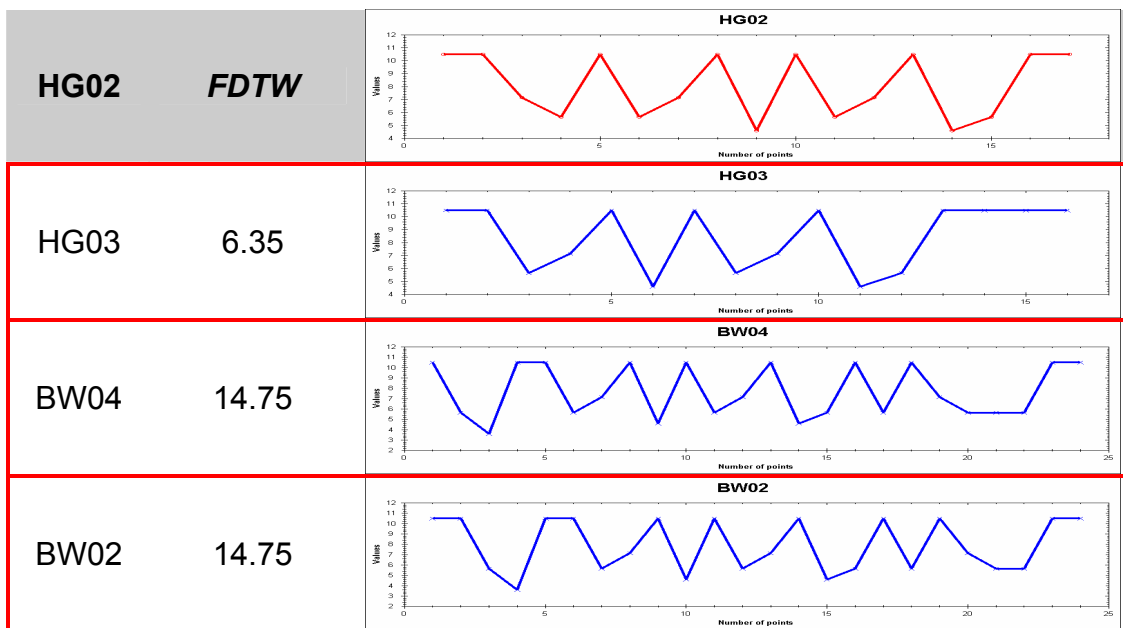
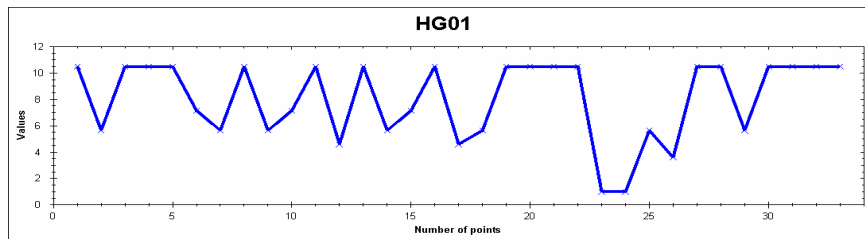


Figura 5.8. Representación gráfica de los tres códigos similares al código de referencia *HG02*.



**Figura 5.9.** Representación gráfica del código *HG01*.

A comparación con las secuencias numéricas similares a *HG02*, presentadas en la Figura 5.8, la distancia *FDTW* de *HG01* con respecto a *HG02*, es mayor a las distancias con respecto a *BW04* y *BW02*. Ver Tabla 5.2.

En la Figura 5.10, se presentan las secuencias numéricas de los tres códigos similares a *Rumelhart02*. En el conjunto de datos, se encuentran dos códigos de la clase *Rumelhart* descrita en el Capítulo 4, es por ello que en segundo y tercer lugar se presentan los códigos de la clase *ED*. Los códigos *Rumelhart01* y *Rumelhart02*, se encuentran incluidos en el CD correspondiente a este trabajo.

En la Figura 5.11, se muestran los códigos similares a *BW01*, en primer y segundo lugar se encuentran los códigos *GS01* y *GS02*, con distancias *FDTW* de 25.65 y 30.10 respectivamente; en tercer lugar se encuentra el código *AMP01* con distancia *FDTW* de 32.40.

Los códigos de la clase *BW*, realizan la conversión de una imagen en escala de grises a una imagen en blanco y negro; las representaciones en secuencia de esta clase “denota” las diferencias entre los códigos que representan. Ver Figura 5.12.

En la Figura 5.13, se muestran las representaciones en secuencia numérica de diversos códigos utilizados en los experimentos.

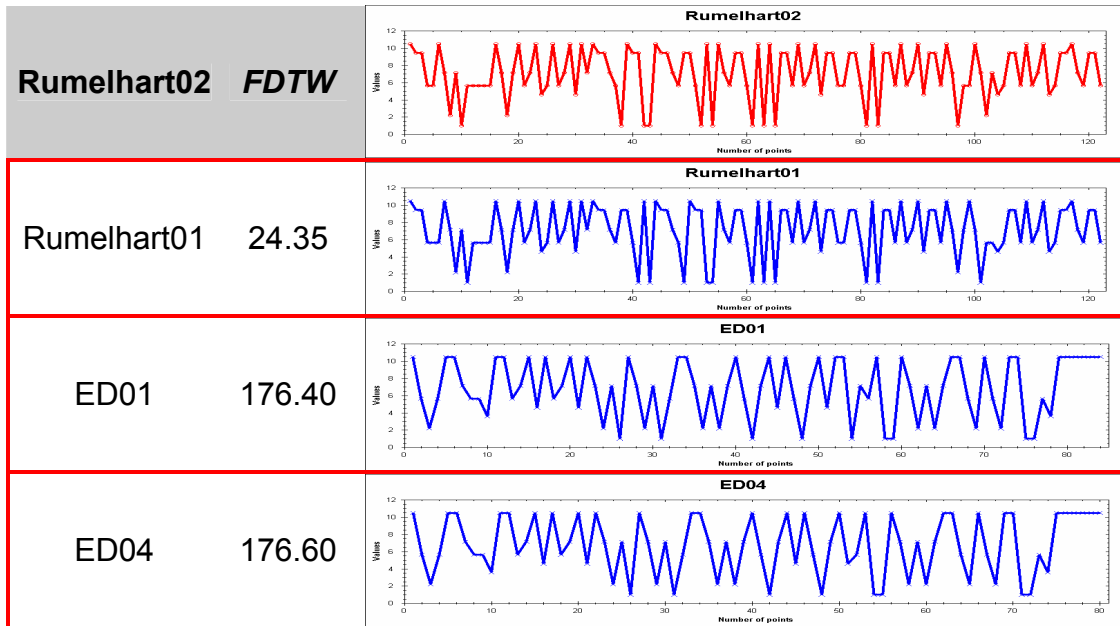


Figura 5.10. Representación gráfica de los tres códigos similares al código de referencia *Rumelhart02*.

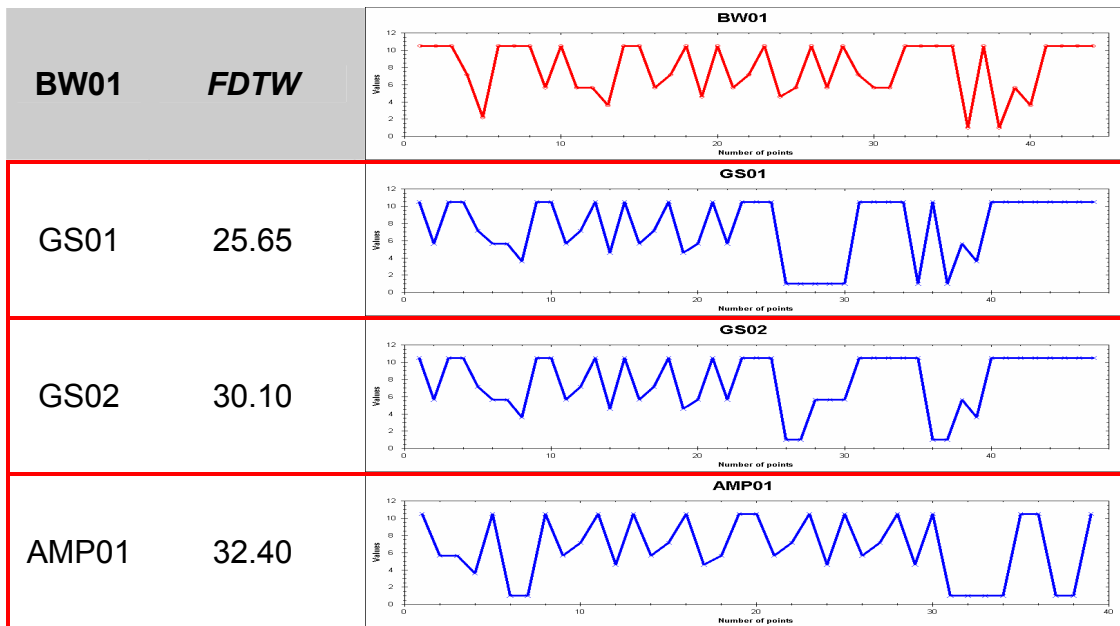


Figura 5.11. Representación gráfica de los tres códigos similares al código de referencia *BW01*.

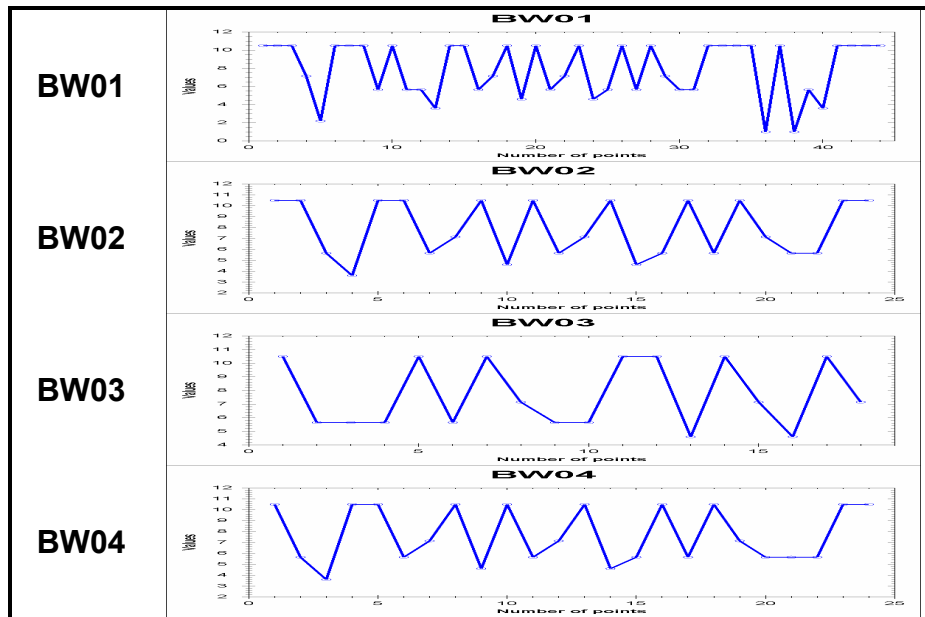


Figura 5.12. Representación gráfica de los códigos pertenecientes a la clase *BW*.

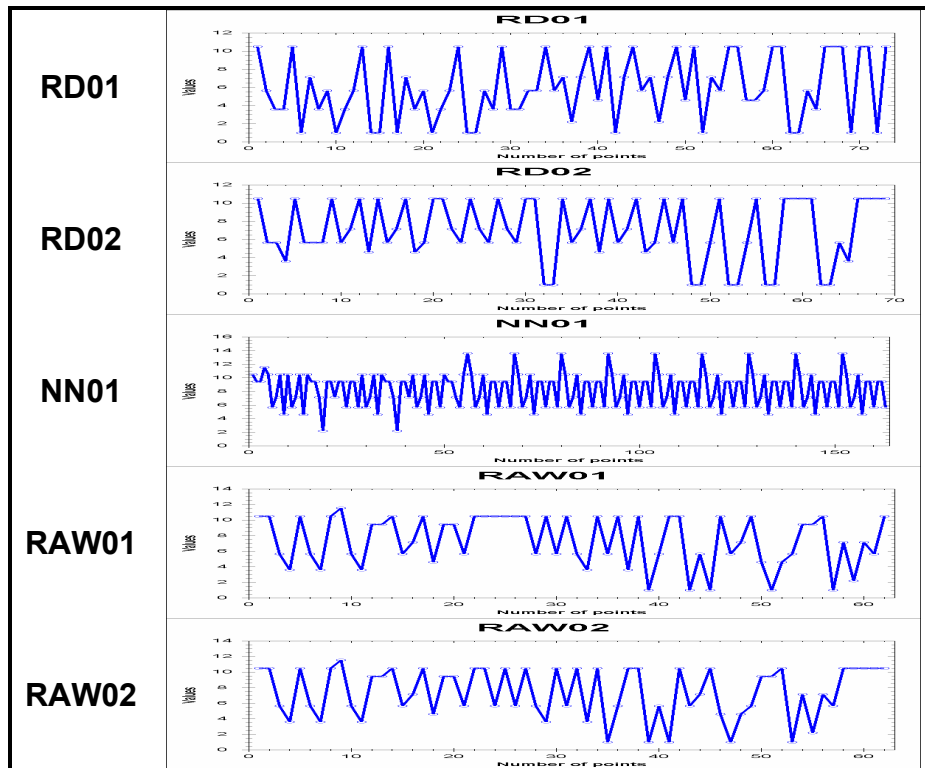


Figura 5.13. Representación gráfica de códigos utilizados en los experimentos.



```

using System;
using System.Net.Sockets;
using System.IO;
public class MyClient{
static void Main(string[] args){
    TcpClient socketForServer; int port = 65432;
    string adress = "localhost";bool status = true;
    try{ //Obtener la ip del equipo donde se ejecuta el cliente
MyClient.exe
        string myip = System.Net.Dns.GetHostEntry(
            System.Net.Dns.GetHostName()).AddressList[0].ToString();
        Console.WriteLine("Mi ip es : " + myip);
        //A donde se va a conectar
        Console.Write("IP del servidor: ");
        adress = Console.ReadLine();
        socketForServer = new TcpClient(adress, port);
        Console.WriteLine("Conectado al servidor");
    }catch{Console.WriteLine("Error al conectar con{0}:{1}",
        adress,port);return;}

    //Los mismos streams que se usan en MyServer
    NetworkStream networkStream = socketForServer.GetStream();
    StreamReader streamreader = new StreamReader(networkStream);
    StreamWriter streamwriter = new StreamWriter(networkStream);
    try{
        string clientmessage = ""; string servermessage = "";
        while (status){
            Console.Write("Cliente: ");
            clientmessage = Console.ReadLine();
            if ((clientmessage == "bye") || (clientmessage == "BYE")){
                status = false;
                streamwriter.WriteLine("bye");
                streamwriter.Flush();
            }if ((clientmessage != "bye") && (clientmessage != "BYE")){
                streamwriter.WriteLine(clientmessage);
                streamwriter.Flush();
                servermessage = streamreader.ReadLine();
                Console.WriteLine("Servidor: " + servermessage);
            }
        }
    }catch{
        Console.WriteLine("Ocurrio un error en el servidor.");
    }
    streamreader.Close();
    networkStream.Close();
    streamwriter.Close();}
}

```

**Figura 5.14.** Código *Cliente01*

```
using System;
using System.Net.Sockets;
using System.IO;
public class Cliente{
static void Main(string[] args){
    int port = 67888; TcpClient socketForServer; string address;
    Console.Write("Conectar a:");
    adress = Console.ReadLine();
    socketForServer = new TcpClient(adress, port);
    Console.WriteLine("Conectado al servidor");
    NetworkStream networkStream = socketForServer.GetStream();
    StreamReader streamreader = new StreamReader(networkStream);
    StreamWriter streamwriter = new StreamWriter(networkStream);
    try{
        string clientmessage = null; string servermessage = null;
        while (clientmessage != "bye"){
            Console.Write("Cliente: ");
            clientmessage = Console.ReadLine();
            if (clientmessage == "bye"){
                status = false;
                streamwriter.WriteLine("bye");
                streamwriter.Flush();
            }if (clientmessage != "bye"){
                streamwriter.WriteLine(clientmessage);
                streamwriter.Flush();
                servermessage = streamreader.ReadLine();
                Console.WriteLine("Servidor: " + servermessage);
            }
        }
    }catch{
        Console.WriteLine("Ocurrio un error en el servidor.");
    }
    streamreader.Close();
    networkStream.Close();
    streamwriter.Close();
}
}
```


**Figura 5.15.** Código *Cliente02*

## Códigos Particulares

A continuación, se presentan algunos códigos con modificaciones especiales como: inversión de instrucciones, intercambio de bloques de código, y la comparación del conjunto de datos con código absurdo.

### Caso 1: Inversión de instrucciones.

En la Figura 5.16, se muestra el código fuente que calcula la distancia euclidiana entre dos valores reales, en la misma figura del lado derecho, se muestra el código invertido línea por línea

<pre>static void Main (string[] args) {     double distance = 0.00;     double q = Convert.ToDouble( args[0] );     double c = Convert.ToDouble( args[1] );     distance = q - c;     distance = Math.Pow(distance, 2);     distance = Math.Sqrt(distance);     Console.WriteLine(distance); }</pre>		<pre>static void Main (string[] args) {     Console.WriteLine(distance);     distance = Math.Sqrt(distance);     distance = Math.Pow(distance, 2);     distance = q - c;     double c = Convert.ToDouble( args[1] );     double q = Convert.ToDouble( args[0] );     double distance = 0.00; }</pre>
--	--	--

**Figura 5.16.** Código *Euclidean* calcula la distancia euclidiana (izq.), *Euclidean inverted* programa invertido (der.)

En este caso, el método propuesto en este trabajo, detecta correctamente la inversión de instrucciones. Dado como *query* el código *Euclidean Inverted*, el primer código similar es *Euclidean* con *FDTW*=14.30, de igual forma, dado el código *Euclidean* como *query*, el primer código similar es *Euclidean Inverted*, con la misma distancia *FDTW*. Los códigos más similares a *Euclidean Inverted*, se muestran en la Tabla 5.3, y las representaciones gráficas en la Figura 5.20.

### Caso 2: Intercambio de bloques de instrucciones.

En la Figura 5.17, se muestra el código fuente *HG01* que recorre una imagen de la clase *Bitmap* de C#, y llama una función que genera un histograma. En la Figura 5.18, se muestra el código *HG01 moved*, donde se realiza el intercambio de un bloque de instrucciones.

```
private void generarHistogramaToolStripMenuItem_Click(object sender,
EventArgs e)
{
    NewImage temp = GetInputImage();

    if (temp != null)
    {
        Bitmap input = temp.input;

        for (int i = 0; i < input.Width; i++)
        {
            for (int j = 0; j < input.Height; j++)
            {
                Color c = input.GetPixel(i, j);
                temp.AgregarAHistograma(c);
            }
            SetProgress((i / input.Width) * 100);
        }
    }

    HistogramView ver = new HistogramView();
    ver.MdiParent = this;

    ver.MakeHistograms(temp.name,temp.HR,temp.HG,temp.HB);
    ver.Show();

    SetProgress(100);
}
}
```

**Figura 5.17.** Código *HG01*. Se remarca con borde, el bloque de instrucciones seleccionadas para mover.

Al realizar la búsqueda de los códigos similares a *HG01 moved*, entre los primeros resultados aparecen los códigos *AMP03*, *AMP04* y *AMP12*; el código *HG01*, aparece en el lugar número 15 con *FDTW* = 42.60. Los resultados se muestran en la Tabla 5.3. Este resultado se debe a que, en el conjunto de datos existen códigos similares a *HG01Moved*, con una distancia *FDTW* menor a la distancia que se obtiene entre *HG01Moved* y *HG01*. La detección de intercambio de bloques de instrucciones, dentro de los códigos fuente a comparar, se encuentra sujeta al resto de los elementos del conjunto de datos. La representación del código *HG01* se muestra en Figura 5.9.

```

private void generarHistogramaToolStripMenuItem_Click(object sender,
EventArgs e)
{
    NewImage temp = GetInputImage();

    HistogramView ver = new HistogramView();
    ver.MdiParent = this;

    ver.MakeHistograms(temp.name,temp.HR,temp.HG,temp.HB);
    ver.Show();
    SetProgress(100);

    if (temp != null)
    {
        Bitmap input = temp.input;

        for (int i = 0; i < input.Width; i++)
        {
            for (int j = 0; j < input.Height; j++)
            {
                Color c = input.GetPixel(i, j);
                temp.AgregarAHistograma(c);
            }
            SetProgress((i / input.Width) * 100);
        }
    }
}

```

**Figura 5.18.** Código *HG01 Moved*. Se remarca con borde, el bloque de instrucciones reubicado.

### Caso 3: Código Absurdo

En la Figura 5.19, se muestra el código fuente *Absurdo*, que contiene instrucciones sin sentido aparente, cabe mencionar que este código *itera* indefinidamente en el primer ciclo de instrucciones.

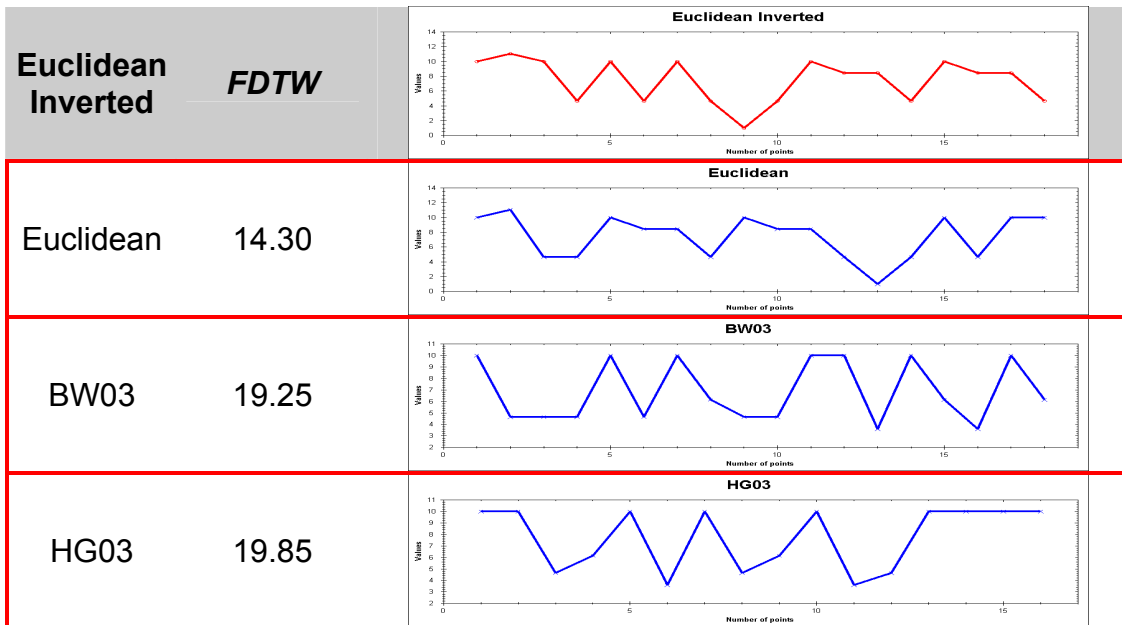
En este experimento, como en todos los experimentos de este trabajo, se obtienen los códigos similares al código de referencia; sin embargo, la distancia del primer código fuente similar a *Absurdo*, es visiblemente superior a cualquier otra distancia *FDTW*, del primer código similar, obtenida en los experimentos. Los resultados se muestran en la Tabla 5.3. En la Figura 5.22, se muestran las representaciones gráficas de los códigos similares a *Absurdo*.

```
static void Main(string[] args){
    while (0 != 1){
        double value = 30 + Math.PI;
        value *= Math.Round(value, 2);
        Console.WriteLine("Forever cicle");
    }
    DirectoryInfo dir = new DirectoryInfo( args[3] );
    foreach (FileInfo file in dir.GetFiles("*.cs"))
    { Console.WriteLine(file as String);}
    for (int i = 20; i = i; i++)
    { Console.Error("Never happends"); }
    double val1 = 20;
    double val2 = 50;
    if (args[0] != null && args[1] != null)
    { int w, h;
      if (val1 > val2) w = val1;
      else w = val1;
      if (val1 > val2) h = val1;
      else h = val1;
    }
    for (int i = 0; i < input.Width; i++)
    for (int j = 0; j < input.Height; j++)
    { Color c = input.GetPixel(i, j);
      if (c.R == 0 && c.G == 0 && c.B == 0)
      { M[i, j] = 0;
        M2[i, j] = 0;
      }
      else
      { M[i, j] = infinito;
        M2[i, j] = infinito;
      }
    }
    for (int i = 0; i < input.Width; i++)
    for (int j = 0; j < input.Height; j++)
    { if (M2[i, j] == infinito)
      pixel = Byte.MaxValue;
      else
      pixel = Convert.ToByte(M2[i, j]);
    bmp.SetPixel(i, j, Color.FromArgb(pixel, pixel, pixel)); }
}
```

**Figura 5.19.** Código *Absurdo*

<b>Euc.Inverted</b>		<b>HG01Moved</b>		<b>Absurdo</b>	
Euc.Inverted	0.00	HG01Moved	0.00	Absurdo	0.00
Euclidean	14.30	AMP03	8.65	RAW01	116.45
BW03	19.25	AMP04	16.65	RAW02	116.45
HG03	19.85	AMP11	23.65	RD02	116.50
HG02	24.90	AMP12	23.65	Cliente01	123.10
AMP08	27.00	AMP02	28.50	ED04	123.35
BW04	33.75	AMP10	28.50	ED01	125.55
BW02	33.75	AMP07	28.50	BW01	127.70
AMP05	38.45	BW03	31.45	ED03	128.45
AMP06	38.45	BW02	34.05	ED02	130.65
...	...	...	...	...	...

**Tabla 5.3.** Comparación de códigos fuente dado un *Query* en el conjunto de datos. Se muestran resaltados con borde rojo, los tres códigos más similares utilizando *FDTW*.



**Figura 5.20.** Representación gráfica de los tres códigos similares al código de referencia *Euclidean Inverted*.

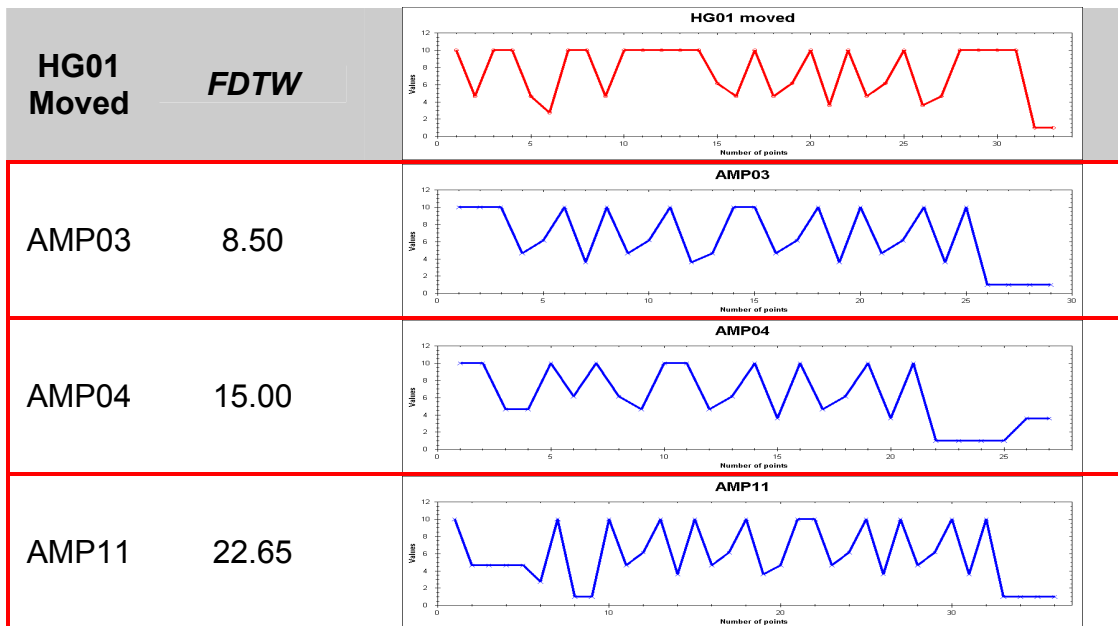


Figura 5.21. Representación gráfica de los tres códigos similares al código de referencia *HG01Moved*.

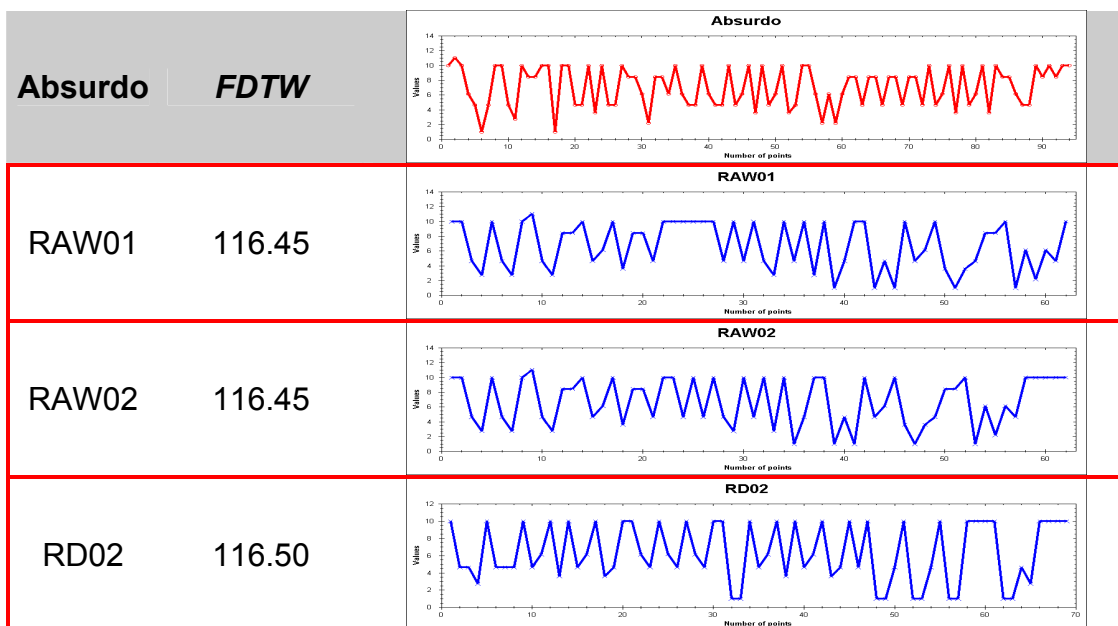


Figura 5.22. Representación gráfica de los tres códigos similares al código de referencia *Absurdo*.



## Valor de Similitud

Al tener un conjunto de códigos fuentes, y calcular la distancia *FDTW* (Capítulo 3) entre dos códigos, se obtiene un valor de similitud entre ellos. Este valor es un escalar que tiende al infinito, adquiere un valor de similitud relativo cuando se computa la distancia *FDTW* entre más de dos códigos fuente, esto es, se obtiene la distancia *FDTW* de cada uno de los elementos que integran el conjunto de datos contra el resto de los elementos.

A continuación se propone un valor de similitud *sim* (Ec. 5.2), que se basa en las distancias *FDTW* entre un conjunto *D* de códigos fuente *d*, donde se obtiene el valor *sim* que proporciona un grado de semejanza normalizado [0,1].

$$D = \{ d_1, d_2, d_s, \dots, d_n \}, \quad n > 2 \quad 5.1$$

$$sim(d_s, d_t, D) = comp\left(\frac{FDTW(d_s, d_t)}{MAX(d_s, D)}\right) \quad 5.2$$

$$MAX(d_s, D) = \max\{FDTW(d_s, d_u)\} \quad 1 \leq u \leq n \quad 5.3$$

$$comp(v) = |v - 1| \quad 5.4$$

Aplicando la medida de similitud propuesta al conjunto de datos, la Tabla 5.1 es equivalente a la Tabla 5.4, donde se obtienen los códigos similares a un código de referencia mostrando el valor *sim* de similitud normalizado entre [0,1].

Es con el valor *sim* de similitud propuesto que las distancias entre: *Server02* y *Server03* es de 0.9524; entre los códigos *ED01* y *ED04* es 0.9706; entre los códigos *AMP04* y *AMP09* es de 0.9743; finalmente entre los códigos *DMA04* y *DMA01* es de 0.8558. Los resultados se muestran en la Tabla 5.4.

<b>Server02</b>		<b>ED01</b>	
Server02	1.0000	ED01	1.0000
Server03	0.9524	ED04	0.9706
Server01	0.9382	ED02	0.9660
Ciente03	0.9341	ED03	0.9367
BW02	0.9197	RD02	0.7884
BW04	0.9197	RAW01	0.7299
HG01	0.9147	RAW02	0.7299
GS01	0.9131	RD01	0.6795
Ciente02	0.9114	Ciente02	0.6749
BW01	0.8987	BW01	0.6668
...	...	...	...

<b>AMP04</b>		<b>DMA04</b>	
AMP04	1.0000	DMA04	1.0000
AMP09	0.9743	DMA01	0.8558
AMP03	0.9686	DMA02	0.8313
AMP05	0.9683	DMA03	0.7684
AMP06	0.9683	Rumelhart02	0.4514
AMP08	0.9427	NN01	0.4448
AMP11	0.9318	Rumelhart01	0.4183
AMP12	0.9318	ED01	0.3367
AMP02	0.9210	ED04	0.3191
AMP07	0.9210	ED02	0.2939
...	...	...	...

**Tabla 5.4.** Representación normalizada de la Tabla 5.1 utilizando el valor *sim*,

Es con la medida propuesta *sim*, que obtenemos un valor normalizado asociable a un grado de semejanza entre códigos fuente.

En el Anexo C muestra el cómputo de similitud para el conjunto de datos de códigos fuente, mostrado en el Capítulo 4, utilizando el valor de similitud propuesto y la codificación abstracta.

## Codificación en Detalle

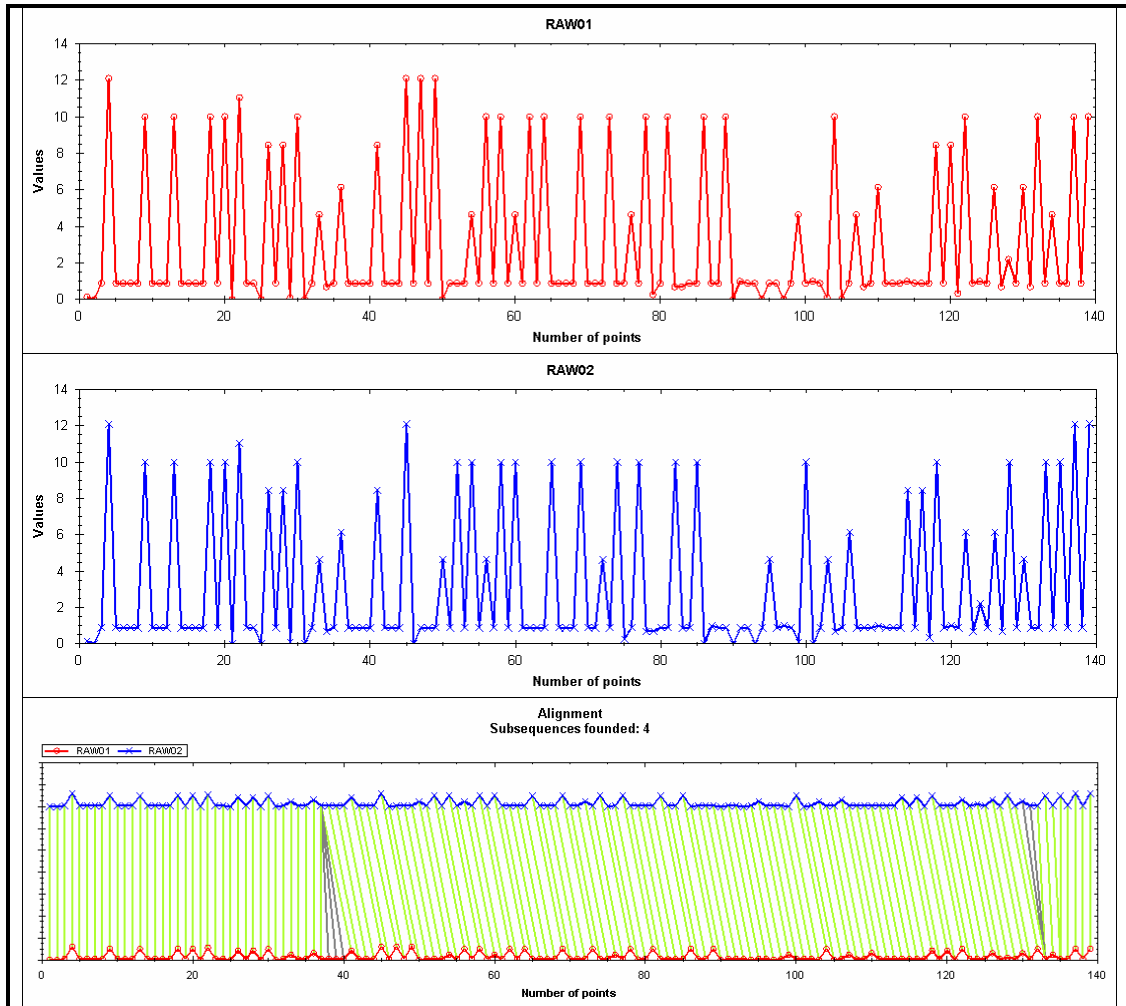
En esta sección, se muestran los resultados obtenidos mediante la transformación del código fuente utilizando la codificación en detalle, descrita en el Capítulo 3, algunos resultados forman parte de [Mirón-Bernal07d].

En la Tabla 5.5, se muestra un resumen comparativo realizado a los códigos *RAW01* y *RAW02*, donde se muestra: la distancia *FDTW* entre las secuencias representativas de estos códigos, utilizando la codificación abstracta; el valor de similitud utilizando la representación mencionada; el valor de la distancia *FDTW* entre las secuencias representativas de estos códigos, utilizando la codificación en detalle; el valor de similitud entre estos códigos utilizando la codificación en detalle.

Resumen ( RAW01 , RAW02 )	
<i>Distancia FDTW (Cod. Abstracta):</i>	<i>0.00</i>
<i>Valor de Similitud :</i>	<i>1.0000</i>
<i>Distancia FDTW (Cod. en Detalle):</i>	<i>44.80</i>
<i>Valor de Similitud :</i>	<i>0.9509</i>

**Tabla 5.5** Resumen comparativo entre los códigos *RAW01* y *RAW02*

En la Figura 5.23, se muestra la representación gráfica de los códigos *RAW01*(rojo) y *RAW02* (azul), utilizando la codificación en detalle. En la parte inferior, se muestra el “alineamiento” (negro) producido por el *warp path*, [Capítulo 3] de la técnica *FDTW*, en color verde se muestran las subsecuencias identificadas por el algoritmo propuesto, los resultados de identificación de subsecuencias se mostrarán en la siguiente sección.

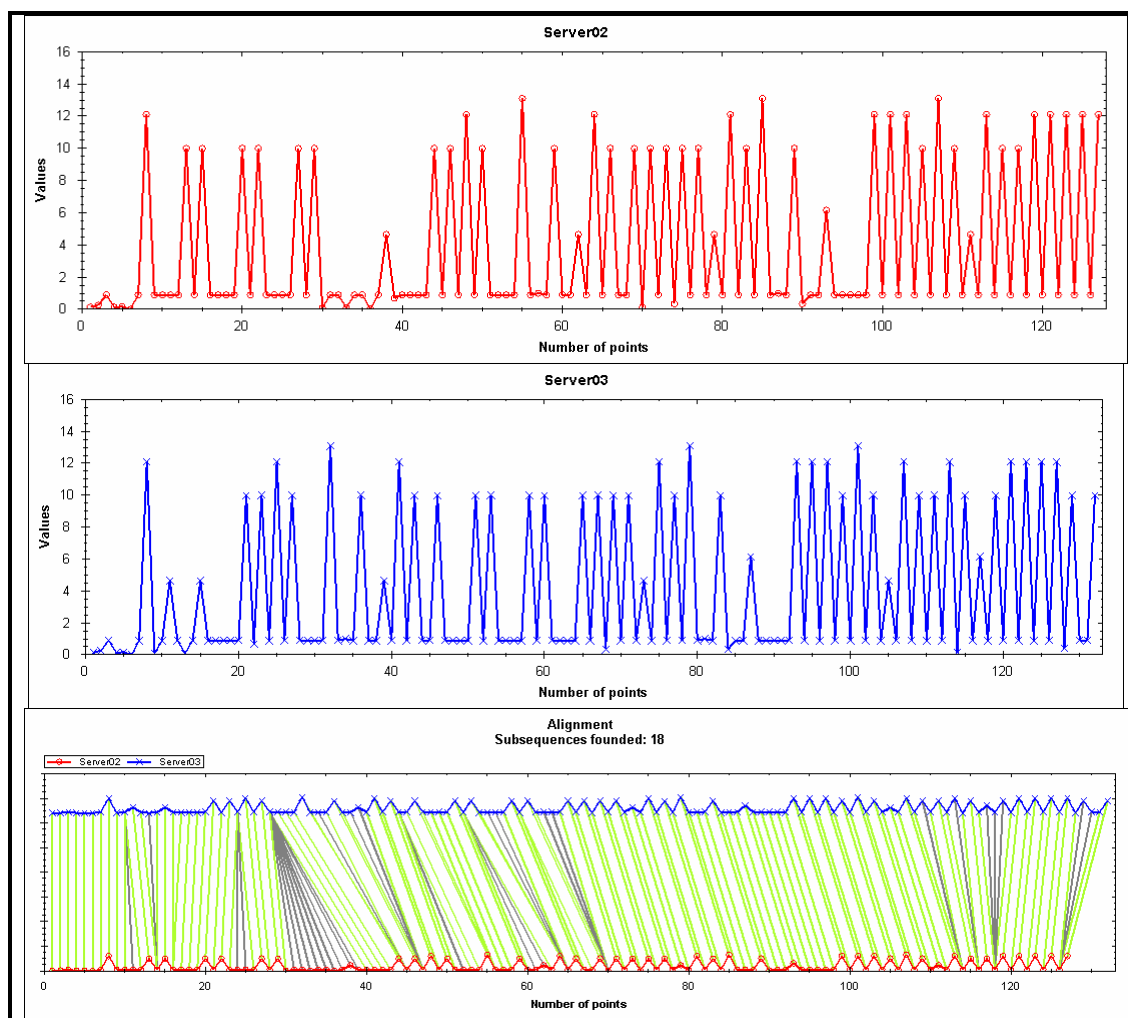


**Figura 5.23.** Representación utilizando la codificación en detalle. En color rojo se muestra el código *RAW01*, en color azul se muestra el código *RAW02*. En la parte inferior se muestra el alineamiento producido por *FDTW*.

Cuando se detecta un valor *sim* cercano al valor máximo (uno), utilizando la codificación abstracta, puede recurrirse a la codificación en detalle para establecer un valor *sim* que incluye más detalles del código. La codificación en detalle se encuentra descrita en el Capítulo 3.

Resumen ( Server02 , Server03 )	
Distancia FDTW (Cod. Abstracta):	23.60
Valor de Similitud :	0.9525
Distancia FDTW (Cod. en Detalle):	87.28
Valor de Similitud :	0.9017

**Tabla 5.6.** Resumen comparativo entre los códigos *Server02* y *Server03*.



**Figura 5.24** Representación utilizando la codificación en detalle. En color rojo se muestra el código *Server02*, en color azul se muestra el código *Server03*. En la parte inferior se muestra el alineamiento producido por *FDTW*.

En la Tabla 5.6, se muestra un resumen comparativo realizado a los códigos *Server02* y *Server03*. Estos códigos tienen como propósito enviar o recibir datos a un programa de la clase *Cliente*, mediante el uso de sockets. El valor de similitud, con la codificación en detalle es de 0.9017. Ver Figura 5.24.

A continuación se presentan los códigos fuente *Cliente03*, *Server02*, y *Server03* en las Figuras 5.25, 5.26, y 5.27 respectivamente.

```
using System;
using System.Net.Sockets;
using System.IO;

public class Client
{
    static void Main(string[] args)
    {
        NetworkStream networkStream = socketForServer.GetStream();
        StreamReader streamreader = new StreamReader(networkStream);
        StreamWriter streamwriter = new StreamWriter(networkStream);
        TcpClient socketForServer;

        socketForServer = new TcpClient("148.204.45.241", 87695);
        string msg_client, msg_server;

        do
        {
            Console.Write("Cliente: ");
            msg_client = Console.ReadLine();
            if (msg_client == "exit")
            {
                status = false;
                streamwriter.WriteLine("bye");
                streamwriter.Flush();
            }
            if (msg_client != "exit")
            {
                streamwriter.WriteLine(msg_client);
                streamwriter.Flush();
                msg_server = streamreader.ReadLine();
                Console.WriteLine("Servidor: " + msg_server);
            }
        } while (msg_client != "exit")

        streamreader.Close();
        networkStream.Close();
        streamwriter.Close();
    }
}
```

**Figura 5.25.** Programa *Cliente03*. Realiza la comunicación mediante sockets con un programa de la clase *Server*.

```
using System;
using System.Net.Sockets;
using System.IO;

public class Servidor
{
    public static void Main()
    {
        NetworkStream networkStream = new NetworkStream(socketForClient);
        StreamWriter streamwriter = new StreamWriter(networkStream);
        StreamReader streamreader = new StreamReader(networkStream);

        bool status = true;
        string servermessage ,clientmessage;

        int port = 67888;

        TcpListener tcpListener = new TcpListener(port);
        tcpListener.Start();

        Console.WriteLine("Inicio del servidor en : " + myip);
        Socket socketForClient = tcpListener.AcceptSocket();
        Console.WriteLine("Cliente Conectado");

        while (status)
        {
            if (socketForClient.Connected)
            {
                //Lee del socket lo que dice el cliente
                servermessage = streamreader.ReadLine();
                Console.WriteLine("Cliente:" + servermessage);
                if ((servermessage == "bye"))
                {
                    status = false;
                    streamreader.Close();
                    networkStream.Close();
                    streamwriter.Close();
                    return;
                }
                //Escribe una respuesta al cliente
                Console.Write("Servidor:");
                clientmessage = Console.ReadLine();
                streamwriter.WriteLine(clientmessage);
                streamwriter.Flush();
            }
            streamreader.Close();
            networkStream.Close();
            streamwriter.Close();
            socketForClient.Close();
        }
    }
}
```

**Figura 5.26.** Programa *Server02*. Realiza la comunicación mediante sockets con un programa de la clase *Cliente*.

```

using System;
using System.Net.Sockets;
using System.IO;
public class MyServer
{
    public static void Main()
    {
        try
        {
            string msg_server = "";
            string msg_client = "";

            TcpListener tcpListener = new TcpListener(87695);
            tcpListener.Start();

            Console.WriteLine("Inicio del servidor en : " + "148.204.45.241" );
            Socket socketForClient = tcpListener.AcceptSocket();
            Console.WriteLine("Cliente Conectado");

            NetworkStream networkStream = new NetworkStream(socketForClient);
            StreamWriter streamwriter = new StreamWriter(networkStream);
            StreamReader streamreader = new StreamReader(networkStream);

            do
            {
                if (socketForClient.Connected)
                {
                    //Lee del socket lo que dice el cliente
                    msg_server = streamreader.ReadLine();
                    Console.WriteLine("Cliente:" + msg_server);
                    if ((msg_server == "bye"))
                    {
                        status = false;
                        streamreader.Close();
                        networkStream.Close();
                        streamwriter.Close();
                        return;
                    }
                    //Escribe una respuesta al cliente
                    Console.Write("Servidor:");
                    msg_client = Console.ReadLine();
                    streamwriter.WriteLine(msg_client);
                    streamwriter.Flush();
                }
            }while(msg_client != "exit");

            streamreader.Close();
            networkStream.Close();
            streamwriter.Close();
            socketForClient.Close();
        }
        catch (Exception e)
        {
        }
    }
}

```

**Figura 5.27.** Programa *Server03*. Realiza la comunicación mediante *sockets* con un programa de la clase *Cliente*.



## Variación de Parámetros

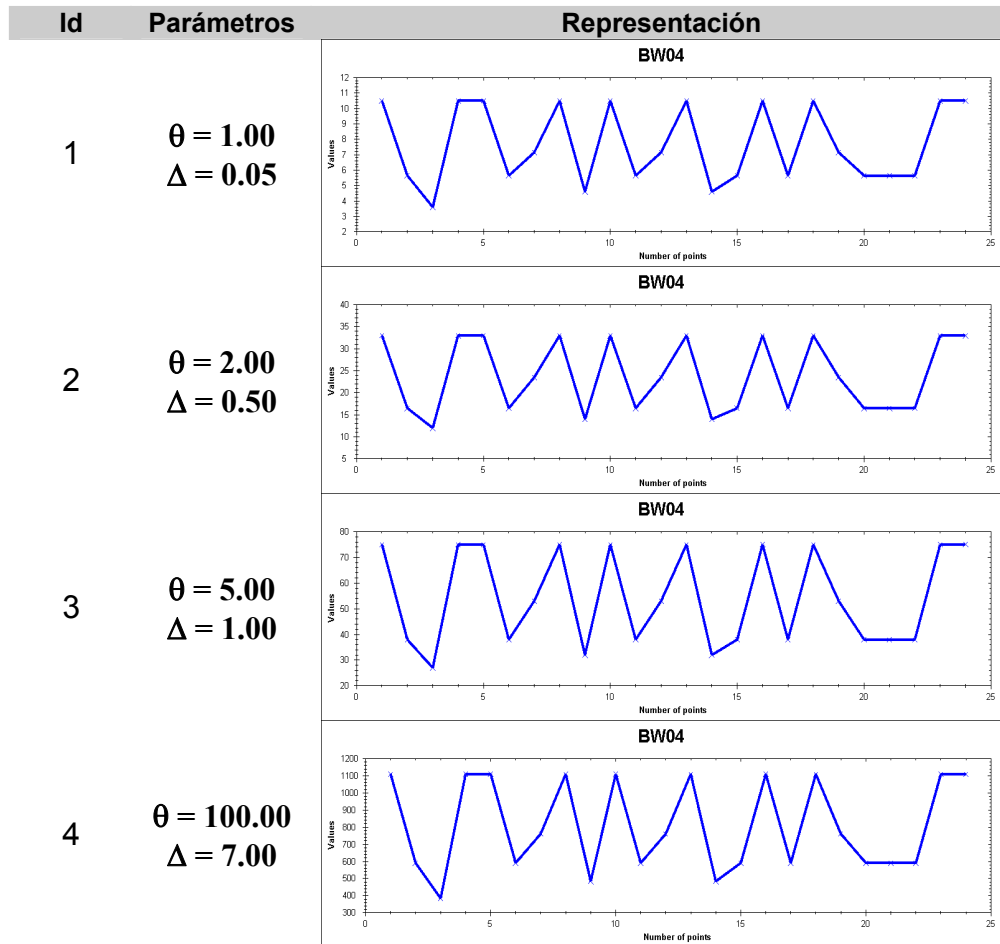
En esta sección, se mostrarán los resultados obtenidos con diferentes valores de codificación. En el Capítulo 3, se explicaron las reglas que permiten llevar a cabo la codificación abstracta y en detalle; ambas codificaciones están sujetas a valores de inicio e incremento respectivamente. Las reglas de codificación no varían.

### Codificación Abstracta

En la Figura 5.28, se muestra la representación del código *BW04*, utilizando diferentes valores para  $\theta$  y  $\Delta$  [Capítulo 3]. Como se observa, la estructura del código fuente se preserva, la principal característica al realizar un cambio de parámetros se muestra en la escala de valores (amplitud); en el primer ejemplo, el eje de valores se encuentra entre [2,12]; en el segundo ejemplo los valores se encuentran entre [5,40]; el intervalo para el tercer ejemplo es [20,80], en el último ejemplo la escala se encuentra entre [300,1200].

Para comparar los resultados obtenidos con parámetros diferentes, utilizaremos las tablas de resultados mostradas en secciones anteriores. Al principio de este Capítulo, se presentaron los códigos similares a *ED01*, *DMA04*, *Server02* y *AMP04*, en la Tabla 5.1, utilizando la codificación abstracta con valores  $\theta = 1.00$  y  $\Delta = 0.05$ .

En las Tablas 5.7, 5.8, 5.9, se muestran comparaciones con la Tabla 5.1, utilizada en secciones anteriores. Los resultados sugieren que al realizar un cambio de valores  $\theta$  y  $\Delta$ , se preserva la estructura del código fuente transformado utilizando la codificación abstracta. Los resultados apoyan la idea propuesta en el Capítulo 3, donde se sustenta la existencia de la estructura dentro del código fuente.



**Figura 5.28.** Cambios de representación al código *BW04*, con diferentes valores  $\theta$  y  $\Delta$ , para la codificación abstracta.

En los experimentos mostrados en las tablas siguientes, se muestra que, dado un código fuente como *query*, en las diferentes codificaciones, se preserva el orden de los códigos más similares a *ED01*, *DMA04* y *Server02*; para *AMP04*, se preserva el primer código similar *AMP09*. Se muestran en las Tablas 5.1, 5.7, 5.8, 5.9, remarcados con borde, los códigos similares que se preservan en orden, aún variando los valores de codificación. En el caso del código *ED01*, en las cuatro variaciones de codificación se muestran invariantes los siete códigos similares, nótese el valor de la distancia *FDTW* en los resultados.

ED01		DMA04		Server02		AMP04	
ED01	0.00	DMA04	0.00	Server02	0.00	AMP04	0.00
ED04	10.50	DMA01	66.95	Server03	23.60	AMP09	12.80
ED02	12.15	DMA02	78.30	Server01	30.65	AMP03	15.65
ED03	22.65	DMA03	107.50	Cliente03	32.70	AMP05	15.80
RD02	75.75	Rumelhart02	254.70	BW02	39.85	AMP06	15.80
RAW01	96.70	NN01	257.75	BW04	39.85	AMP08	28.60
RAW02	96.70	Rumelhart01	270.05	HG01	42.35	AMP11	34.05
RD01	114.75	ED01	307.95	GS01	43.10	AMP12	34.05
Cliente02	116.40	ED04	316.15	Cliente02	43.95	AMP02	39.40
BW01	119.30	ED02	327.80	BW01	50.25	AMP07	39.40
...	...	...	...	...	...	...	...

Tabla 5.1. Repetición Tabla 5.1. Tabla de resultados con valores  $\theta = 1.00$  y  $\Delta = 0.05$ .

ED01		DMA04		Server02		AMP04	
ED01	0.00	DMA04	0.00	Server02	0.00	AMP04	0.00
ED04	38.00	DMA01	221.50	Server03	72.00	AMP09	38.00
ED02	47.50	DMA02	255.00	Server01	94.50	AMP05	52.00
ED03	85.50	DMA03	340.50	Cliente03	119.50	AMP06	52.00
RD02	294.50	NN01	772.50	BW02	129.00	AMP03	59.50
RAW02	359.50	Rumelhart02	807.50	BW04	129.00	AMP08	90.00
RAW01	359.50	Rumelhart01	845.50	Cliente02	136.00	AMP11	119.00
GS02	428.50	ED01	957.50	HG01	136.00	AMP12	119.00
Cliente02	431.50	ED04	986.50	GS01	139.00	HG01	133.00
Cliente01	433.50	ED02	1024.00	BW03	169.00	AMP07	135.50
...	...	...	...	...	...	...	...

Tabla 5.7. Tabla de resultados con valores  $\theta = 2.00$  y  $\Delta = 0.50$

ED01		DMA04		Server02		AMP04	
ED01	0.00	DMA04	0.00	Server02	0.00	AMP04	0.00
ED04	85.00	DMA01	501.00	Server03	168.00	AMP09	86.00
ED02	108.00	DMA02	578.00	Server01	219.00	AMP05	116.00
ED03	193.00	DMA03	769.00	Cliente03	272.00	AMP06	116.00
RD02	666.00	NN01	1770.00	BW02	296.00	AMP03	133.00
RAW02	809.00	Rumelhart02	1827.00	BW04	296.00	AMP08	202.00
RAW01	809.00	Rumelhart01	1915.00	HG01	311.00	AMP11	269.00
GS02	963.00	ED01	2169.00	Cliente02	314.00	AMP12	269.00
Cliente02	971.00	ED04	2232.00	GS01	319.00	HG01	303.00
Cliente01	977.00	ED02	2321.00	BW03	388.00	AMP10	306.00
...	...	...	...	...	...	...	...

Tabla 5.8. Tabla de resultados con valores  $\theta = 5.00$  y  $\Delta = 1.00$

ED01		DMA04		Server02		AMP04	
ED01	0	DMA04	0	Server02	0	AMP04	0
ED04	1180	DMA01	7277	Server03	2736	AMP09	1252
ED02	1601	DMA02	8466	Server01	3483	AMP06	1592
ED03	2781	DMA03	11103	Cliente03	4049	AMP05	1592
RD02	9586	Rumelhart02	26569	BW04	4542	AMP03	1841
RAW02	11513	NN01	27015	BW02	4542	AMP08	2844
RAW01	11513	Rumelhart01	27912	HG01	4712	AMP12	3898
GS02	13631	ED01	31677	Cliente02	4854	AMP11	3898
Cliente02	13817	ED04	32415	GS01	4898	AMP02	4417
BW01	13924	ED02	33976	BW01	5725	AMP07	4417
...	...	...	...	...	...	...	...

**Tabla 5.9.** Tabla de resultados con valores  $\theta = 100.00$  y  $\Delta = 7.00$

Para el código *DMA04*, se preservan los primeros cuatro códigos similares en las diferentes codificaciones, además el lector puede observar, en la Tabla 5.1 y 5.9, con respecto al código de referencia *DMA04*, el código *Rumelhart02* obtiene una distancia *FDTW* menor que el código *NN01*, sin embargo, en la Tabla 5.7 y 5.8, la distancia es mayor. Los resultados sugieren que los valores para la codificación abstracta, pueden “acercar” o “alejarse” una clase determinada de operadores, con respecto a un código de referencia, sin embargo, los primeros lugares preservan el orden en las diferentes codificaciones.

Tomando como referencia el código *Server02*, los primeros seis códigos similares se preservan en las diferentes codificaciones, el lector puede observar que los últimos códigos mostrados en las tablas, incluyen códigos de las clases *HG*, *Cliente*, *GS* y *BW*; los resultados no incluyen códigos de diferentes clases, los datos sugieren relación del código *AMP04* con las clases mencionadas anteriormente.

En los resultados mostrados, para el código *AMP04*, sólo se preserva el orden de los primeros dos códigos similares, cabe mencionar que los códigos *AMP03*, *AMP05*, *AMP06*, se encuentran presentes en las diferentes codificaciones después de *AMP09*, variando el orden de presentación, sin embargo los códigos *AMP06* y *AMP05*, obtienen la misma distancia con respecto a *AMP09*, en las diferentes codificaciones.

## Codificación en Detalle

A continuación, se muestran resultados utilizando la codificación en detalle con diferentes valores para las variables, constantes y  $\sigma$ . La descripción de estos valores, se encuentra en el Capítulo 3. Se utiliza la clasificación propuesta por Microsoft®, también descrita en el Capítulo 3. Las reglas de codificación no varían. Los valores para  $\sigma = 0.00625$ ,  $\sigma = 0.0125$ ,  $\sigma = 0.05$ , son el resultado de la división entre 0.5, 1, 4, y el número de palabras reservadas respectivamente [80]; al numerar primero las palabras reservadas se obtuvo  $\sigma = 1.00$ ; con estos valores, se cumplen las convenciones especificadas en el Capítulo 3: no debe existir intersección entre los valores asignados a: las palabras reservadas, los operadores, las variables y constantes.

En las Tabla 5.10, 5.11, 5.12, 5.13, se muestran remarcados con borde, los códigos que preservan el orden de semejanza con respecto al código de referencia, a través de las diferentes codificaciones.

RAW01		AMP08		GS02		Server03	
RAW01	0.00	AMP08	0.00	GS02	0.00	Server03	0.00
RAW02	46.80	AMP09	21.73	GS01	36.16	Server02	92.39
GS01	97.97	AMP04	51.05	BW01	71.76	Server01	98.15
Ciente03	98.28	AMP06	52.75	RD01	83.91	Ciente02	107.45
GS02	103.83	AMP05	52.75	RAW02	98.28	Ciente03	121.82
BW01	105.51	BW04	59.91	Ciente03	101.47	GS02	135.94
RD01	122.32	HG03	62.00	RAW01	103.83	RAW01	135.97
Server02	124.93	AMP03	63.70	Server02	105.84	RAW02	138.13
RD02	125.64	HG02	64.94	RD02	107.83	RD02	138.24
Server03	135.97	BW02	66.16	HG01	135.80	RD01	151.09
...	...	...	...	...	...	...	...

**Tabla 5.10.** Tabla de resultados con valores  $\theta = 1.00$ ,  $\Delta = 0.05$ ,  $\sigma = 0.00625$ , constantes = 0.7, variables = 0.9 para la codificación en detalle.

RAW01		AMP08		GS02		Server03	
RAW01	0.00	AMP08	0.00	GS02	0.00	Server03	0.00
RAW02	146.40	AMP09	66.25	GS01	114.33	Server02	255.30
Cliente03	257.03	AMP04	162.05	BW01	216.35	Cliente02	294.96
GS01	281.78	BW04	163.43	RD01	260.63	Server01	306.00
GS02	283.43	AMP06	175.90	Cliente03	270.68	Cliente03	342.85
BW01	306.23	AMP05	175.90	RAW02	274.99	RAW01	367.96
Server02	328.25	BW02	188.71	Server02	276.49	RD02	374.50
RD01	353.11	HG03	189.60	RAW01	283.43	GS02	377.80
Server03	367.96	AMP02	189.96	RD02	321.35	RAW02	378.48
RD02	375.46	AMP10	189.96	Server03	377.80	RD01	426.04
...	...	...	...	...	...	...	...

**Tabla 5.11.** Tabla de resultados con valores  $\theta = 2.00$ ,  $\Delta = 0.50$ ,  $\sigma = 0.0125$ , constantes = 1.8, variables = 1.4 para la codificación en detalle.

RAW01		AMP08		GS02		Server03	
RAW01	0.00	AMP08	0.00	GS02	0.00	Server03	0.00
RAW02	330.40	AMP09	150.60	GS01	257.70	Server02	593.40
Cliente03	604.10	AMP04	362.00	BW01	484.20	Cliente02	685.25
GS01	633.10	BW04	372.90	RD01	570.10	Server01	690.00
GS02	646.10	AMP06	392.40	Cliente03	627.10	Cliente03	788.20
BW01	692.50	AMP05	392.40	RAW02	628.75	RAW01	855.45
Server02	771.20	BW02	426.05	Server02	644.75	RD02	864.60
RD01	792.25	HG03	427.40	RAW01	646.10	GS02	869.00
RD02	848.55	AMP02	438.65	RD02	721.25	RAW02	876.70
Server03	855.45	AMP10	438.65	Server03	869.00	RD01	963.15
...	...	...	...	...	...	...	...

**Tabla 5.12.** Tabla de resultados con valores  $\theta = 5.00$ ,  $\Delta = 1.00$ ,  $\sigma = 0.05$ , constantes= 4.6, variables = 4.4 para la codificación en detalle.

RAW01		AMP08		GS02		Server03	
RAW01	0	AMP08	0	GS02	0	Server03	0
RAW02	4936	AMP09	2240	GS01	3820	Server02	9580
GS01	9874	AMP04	5314	BW01	7300	Server01	10283
Cliente03	9974	AMP05	5604	RD01	8425	Cliente02	10936
GS02	10375	AMP06	5604	RAW02	9884	Cliente03	12536
BW01	10671	BW04	5897	Cliente03	10240	GS02	13827
RD01	12271	HG03	6430	RAW01	10375	RAW01	13844
Server02	12679	BW02	6592	Server02	10632	RD02	13889
RD02	12860	AMP03	6675	RD02	10856	RAW02	14083
Server03	13844	HG02	6796	Server03	13827	RD01	15108
...	...	...	...	...	...	...	...

**Tabla 5.13.** Tabla de resultados con valores  $\theta = 100.00$ ,  $\Delta = 7.0$ ,  $\sigma = 1.00$ , constantes = 85.00, variables = 90.00 para la codificación en detalle.

Tomando como código de referencia *RAW01* [código que convierte un archivo binario en imagen], en las diferentes codificaciones aparece en primer lugar el código *RAW02*, posteriormente los códigos *GS01* y *Cliente03*, muestran ser el tercer y cuarto código similar con respecto a *RAW01*, en las Tablas 5.10, 5.11, y 5.12, ambos códigos presentan distancia *FDTW* diferente con respecto a las diferentes codificaciones, nótese en la Tabla 5.13, estos códigos se presentan con la misma distancia con respecto a *RAW01*. En este experimento, también se preservan los códigos similares al código de referencia, cambiando únicamente el orden de presentación.

Para el código *AMP08*, se preserva el orden de los tres primeros códigos similares incluyendo a *AMP09* y *AMP04*. Los códigos *AMP05* y *AMP06*, preservan su distancia con respecto a *AMP08* en las diferentes codificaciones.

Tomando como referencia el código *GS02*, se preservan los primeros cuatro códigos similares a través de las diferentes codificaciones, incluyendo a *GS01*, *BW01*, *RD01*; posteriormente, se presentan los códigos *RAW02*, *Cliente03*, *RAW01*, *Server02*, en diferente orden de presentación según la codificación; esto se debe, a la estructura que preservan en relación con *GS02*.

Finalmente para el código *Server03*, se preserva el código similar *Server02* en las diferentes codificaciones, los códigos *Server01* y *Cliente02*, preservan la estructura que tiene relación con el código de referencia, y se presentan en segundo y tercer lugar dependiendo de la codificación.

En este apartado, se muestra que la variación de parámetros no afecta al primer código similar, dado un código de referencia; los códigos similares siguientes, preservan la relación con el código de referencia, variando el orden de presentación.

## Variaciones en la clasificación de operadores

En este apartado, se muestran los resultados obtenidos utilizando clasificaciones diferentes a la presentada anteriormente en el Capítulo 3. Las reglas de codificación se preservan. En la sección anterior, se utilizó la clasificación presentada por Microsoft® en su documentación [MSDN05], en este apartado, se presentan dos clasificaciones diferentes y los resultados obtenidos con diferentes valores para la codificación.

Para obtener una segunda clasificación, se preservan las categorías de los operadores de la clasificación de Microsoft®, pero se modifica el orden de las clases para su codificación. Las variaciones son presentadas en la Figura 5.29.

<b>Categoría de Operadores</b>	<b>Operadores</b>
<i>Lógicos</i>	<b>&amp;   ^ ! ~ &amp;&amp;    true false</b>
<i>Creación de Objetos</i>	<b>new</b>
<i>Incremento / Decremento</i>	<b>++ --</b>
<i>Aritméticos</i>	<b>+ - * / %</b>
<i>De Cambio</i>	<b>&lt;&lt; &gt;&gt;</b>
<i>Indexado</i>	<b>[ ]</b>
<i>Asignación</i>	<b>= += -= *= /= % &amp;=  = ^= &lt;&lt;= &gt;&gt;=</b>
<i>Control de errores</i>	<b>checked unchecked</b>
<i>Información de Tipo</i>	<b>as is sizeof typeof</b>
<i>Relacionales</i>	<b>== != &lt; &gt; &lt;= &gt;=</b>
<i>Cast</i>	<b>( )</b>

**Figura 5.29.** Segunda clasificación de operadores, variando el orden de codificación de las categorías.



En las Tabla 5.14 y 5.15 se presentan los resultados obtenidos con la segunda clasificación, el lector no debe confundirse, los códigos fuente transformados con diferentes clasificaciones no deben compararse, ya que su comparación puede carecer de significado; de llevarse a cabo, sería semejante a comparar dos códigos realizados en diferentes lenguajes de programación, con reglas de codificación de un tercer lenguaje de programación.

ED01		DMA04		Server02		AMP04	
ED01	0.00	DMA04	0.00	Server02	0.00	AMP04	0.00
ED04	6.9.0	DMA01	61.75	Server03	23.10	AMP06	10.80
ED02	20.10	DMA02	69.70	Server01	24.90	AMP05	10.80
ED03	27.00	DMA03	74.80	Cliente03	40.95	AMP09	13.80
RD01	100.20	ED01	192.65	Cliente02	43.80	AMP03	14.85
RD02	100.20	ED04	197.45	GS02	46.40	AMP08	24.60
RAW01	104.95	ED02	200.55	BW01	46.50	HG02	25.20
RAW02	104.95	ED03	205.35	GS01	48.65	AMP02	26.95
BW01	118.05	Rumelhart01	207.55	BW04	51.15	AMP11	26.95
AMP01	119.35	Rumelhart02	209.55	BW02	51.15	AMP12	26.95
...	...	...	...	...	...	...	...

**Tabla 5.14.** Resultados. Segunda Clasificación. Codificación Abstracta, con valores  $\theta = 1.00$  y  $\Delta = 0.05$ .

ED01		DMA04		Server02		AMP04	
ED01	0.00	DMA04	0.00	Server02	0.00	AMP04	0.00
ED04	21.00	DMA01	177.50	Server03	71.00	AMP05	28.00
ED02	57.00	DMA02	201.00	Server01	73.00	AMP06	28.00
ED03	78.00	DMA03	212.00	Cliente03	121.5	AMP03	38.00
RD02	306.00	ED01	558.50	Cliente02	126.00	AMP09	42.00
RD01	314.00	ED02	573.50	BW02	143.50	HG02	68.00
RAW01	321.50	ED04	574.50	BW04	143.50	AMP12	69.50
RAW02	321.50	ED03	589.50	GS02	144.00	AMP11	69.50
AMP01	361.50	Rumelhart01	608.00	BW01	145.00	AMP07	69.50
BW01	364.50	Rumelhart02	616.50	GS01	150.50	AMP02	69.50
...	...	...	...	...	...	...	...

**Tabla 5.15.** Resultados. Segunda Clasificación. Codificación Abstracta, con valores  $\theta = 2.00$ ,  $\Delta = 0.50$ .

Como se muestra en las Tablas 5.14 y 5.15, se muestran marcados con borde rojo los primeros códigos similares a los códigos de referencia, que coinciden con la Tabla 5.1. mostrada al principio de este Capítulo. Además, se muestra el código de referencia *AMP04*, y sus resultados correspondientes ya que no presentan similitud con la Tabla 5.1 en los primeros códigos similares, y no se remarcan debido al orden en que se presentan utilizando la segunda clasificación; sin embargo, se presentan los mismos códigos *AMP05*, *AMP06*, *AMP09* y *AMP03*, en las diferentes clasificaciones, con variaciones de parámetros como los códigos similares a *AMP04*.

Para obtener una tercera clasificación, se preservan las categorías de los operadores de la clasificación de Microsoft®, modificando el orden de las clases para su codificación; además, se modifica el orden de los operadores que intervienen en cada categoría de operadores. Las variaciones son presentadas en la Figura 5.30.

<b>Categoría de Operadores</b>	<b>Operador</b>
<i>Indexado</i>	
<i>Creación de Objetos</i>	<b>New</b>
<i>Información de Tipo</i>	<b>is as typeof sizeof</b>
<i>Relacionales</i>	<= == < != > >=
<i>Aritméticos</i>	- * + % /
<i>De Cambio</i>	>> <<
<i>Lógicos</i>	<b>true   ^ &amp; ~    &amp;&amp; ! false</b>
<i>Cast</i>	) (
<i>Control de errores</i>	<b>unchecked checked</b>
<i>Asignación</i>	<b>&amp;= &gt;&gt;= ^= &lt;&lt;= -=</b> <b>% = += *=  = /= =</b>
<i>Incremento / Decremento</i>	-- ++

**Figura 5.30.** Tercera clasificación de operadores, variando el orden de codificación de las categorías y el orden de codificación de los operadores dentro de cada categoría.

En las Tablas 5.16 y 5.17, se muestran los resultados comparados con la Tabla 5.1, mostrada al principio de este Capítulo, se preservan los primeros códigos similares al código de referencia, aún variando la clasificación y los valores de codificación. Se muestra el código de referencia *AMP04*, y sus resultados correspondientes ya que no presentan similitud con la Tabla 5.1 en los primeros códigos similares, y no se remarcan debido al orden en que se presentan utilizando la segunda y tercera clasificación; sin embargo, se presentan los mismos códigos *AMP05*, *AMP06*, *AMP09* y *AMP03*, se presentan en las diferentes clasificaciones, con variaciones de parámetros como los códigos similares a *AMP04*.

ED01		DMA04		Server02		AMP04	
ED01	0.00	DMA04	0.00	Server02	0.00	AMP04	0.00
ED04	12.80	DMA01	73.60	Server03	23.15	AMP05	7.20
ED02	13.35	DMA02	85.25	Server01	27.00	AMP06	7.20
ED03	26.15	DMA03	103.00	Cliente03	31.75	AMP09	19.00
RD02	60.85	Rumelhart01	263.20	GS01	35.45	AMP03	23.70
RD01	93.45	NN01	264.75	GS02	36.25	BW02	25.25
AMP01	100.80	Rumelhart02	268.70	BW01	38.05	AMP08	26.20
GS01	101.95	RD01	369.45	HG01	40.60	BW04	26.75
RAW02	105.95	ED02	375.30	BW04	41.15	HG02	33.75
RAW01	105.95	ED03	385.70	BW02	41.15	HG01	33.80
...	...	...	...	...	...	...	...

**Tabla 5.16.** Resultados. Tercera Clasificación. Codificación Abstracta , con valores  $\theta = 1.00$  y  $\Delta = 0.05$ .

ED01		DMA04		Server02		AMP04	
ED01	0.00	DMA04	0.00	Server02	0.00	AMP04	0.00
ED02	37.50	DMA01	208.00	Server03	71.50	AMP06	24.00
ED04	40.00	DMA02	244.50	Server01	78.00	AMP05	24.00
ED03	77.50	DMA03	294.00	Cliente03	93.50	AMP09	62.00
RD02	184.50	NN01	735.50	GS02	106.50	AMP03	77.00
RD01	278.50	Rumelhart01	750.50	GS01	106.50	BW02	84.50
GS01	307.50	Rumelhart02	770.50	BW01	116.50	AMP08	86.00
RAW01	312.00	RD01	1070.50	HG01	118.00	BW04	91.50
RAW02	312.00	ED02	1080.00	BW04	123.50	HG02	113.50
AMP01	312.00	ED03	1105.00	BW02	123.50	HG01	114.00
...	...	...	...	...	...	...	...

**Tabla 5.17.** Resultados. Tercera Clasificación. Codificación Abstracta , con valores  $\theta = 2.00$  y  $\Delta = 0.50$ .

## Comparación de Resultados

En este apartado, se muestra la comparación de los resultados obtenidos en las tres clasificaciones utilizadas con diferentes valores de parámetros, para la codificación abstracta y en detalle.

### Codificación Abstracta

A continuación, en las Tablas 5.18, 5.19 y 5.20, se presenta la comparación de resultados a través de las tres variantes en la clasificación de operadores, con diferentes valores para la codificación abstracta.

En las Tablas 5.18, 5.19 y 5.20, se muestra con bordes en rojo los resultados que preservan el orden de similitud con respecto al código de referencia, a través de las diferentes clasificaciones.

Para los códigos *ED01*, *DMA04* y *Server02*, los tres primeros códigos similares persisten, sin embargo, para el código *AMP04*, en la Tabla 5.18 con la clasificación de Microsoft, los tres primeros códigos similares son *AMP09*, *AMP03*, y *AMP05*, mientras que en la Tabla 5.19 y 5.20, se presentan estos resultados con diferente orden de aparición; esto es, la relación de semejanza de los códigos mencionados con respecto al código *AMP04*, se preserva a través de las diferentes clasificaciones con diferentes valores para la codificación abstracta.

ED01		DMA04		Server02		AMP04	
ED01	0.00	DMA04	0.00	Server02	0.00	AMP04	0.00
ED04	10.50	DMA01	66.95	Server03	23.60	AMP09	12.80
ED02	12.15	DMA02	78.30	Server01	30.65	AMP03	15.65
ED03	22.65	DMA03	107.50	Cliente03	32.70	AMP05	15.80
RD02	75.75	Rumelhart02	254.70	BW02	39.85	AMP06	15.80
RAW01	96.70	NN01	257.75	BW04	39.85	AMP08	28.60
RAW02	96.70	Rumelhart01	270.05	HG01	42.35	AMP11	34.05
RD01	114.75	ED01	307.95	GS01	43.10	AMP12	34.05
Cliente02	116.40	ED04	316.15	Cliente02	43.95	AMP02	39.40
BW01	119.30	ED02	327.80	BW01	50.25	AMP07	39.40
...	...	...	...	...	...	...	...

**Tabla 5.18.** Resultados. Clasificación Microsoft. Codificación Abstracta, con valores  $\theta = 1.00$  y  $\Delta = 0.05$ .

ED01		DMA04		Server02		AMP04	
ED01	0.00	DMA04	0.00	Server02	0.00	AMP04	0.00
ED04	6.9.0	DMA01	61.75	Server03	23.10	AMP06	10.80
ED02	20.10	DMA02	69.70	Server01	24.90	AMP05	10.80
ED03	27.00	DMA03	74.80	Cliente03	40.95	AMP09	13.80
RD01	100.20	ED01	192.65	Cliente02	43.80	AMP03	14.85
RD02	100.20	ED04	197.45	GS02	46.40	AMP08	24.60
RAW01	104.95	ED02	200.55	BW01	46.50	HG02	25.20
RAW02	104.95	ED03	205.35	GS01	48.65	AMP02	26.95
BW01	118.05	Rumelhart01	207.55	BW04	51.15	AMP11	26.95
AMP01	119.35	Rumelhart02	209.55	BW02	51.15	AMP12	26.95
...	...	...	...	...	...	...	...

**Tabla 5.19.** Resultados. Segunda Clasificación. Codificación Abstracta, con valores  $\theta = 1.00$  y  $\Delta = 0.05$ .

ED01		DMA04		Server02		AMP04	
ED01	0.00	DMA04	0.00	Server02	0.00	AMP04	0.00
ED04	12.80	DMA01	73.60	Server03	23.15	AMP05	7.20
ED02	13.35	DMA02	85.25	Server01	27.00	AMP06	7.20
ED03	26.15	DMA03	103.00	Cliente03	31.75	AMP09	19.00
RD02	60.85	Rumelhart01	263.20	GS01	35.45	AMP03	23.70
RD01	93.45	NN01	264.75	GS02	36.25	BW02	25.25
AMP01	100.80	Rumelhart02	268.70	BW01	38.05	AMP08	26.20
GS01	101.95	RD01	369.45	HG01	40.60	BW04	26.75
RAW02	105.95	ED02	375.30	BW04	41.15	HG02	33.75
RAW01	105.95	ED03	385.70	BW02	41.15	HG01	33.80
...	...	...	...	...	...	...	...

**Tabla 5.20.** Resultados. Tercera Clasificación. Codificación Abstracta, con valores  $\theta = 1.00$  y  $\Delta = 0.05$ .

En las Tablas 5.21, 5.22 y 5.23, se muestran los resultados para los códigos de referencia *ED01*, *DMA04*, *Server02* y *AMP04*, con valores para la codificación abstracta  $\theta = 2.00$  y  $\Delta = 0.50$ . El código de referencia *ED01*, se presentan los tres códigos similares *ED04*, *ED02* y *ED03*, en primer lugar en las Tablas 5.21 y 5.22, sin embargo, no se encuentran marcadas en rojo debido a que en la Tabla 5.23, se presentan en diferente orden, no obstante, se encuentran dentro de los primeros códigos similares a *ED01*. De forma similar, los códigos *AMP09*, *AMP05* y *AMP06*, se presentan en diferente orden, a través de las diferentes clasificaciones, pero conservan los tres primeros lugares como códigos similares a *AMP04*. (la distancia *FDTW* con respecto a *AMP04*, de los códigos *AMP05* y *AMP06* es idéntica.)

ED01		DMA04		Server02		AMP04	
ED01	0.00	DMA04	0.00	Server02	0.00	AMP04	0.00
ED04	38.00	DMA01	221.50	Server03	72.00	AMP09	38.00
ED02	47.50	DMA02	255.00	Server01	94.50	AMP05	52.00
ED03	85.50	DMA03	340.50	Cliente03	119.50	AMP06	52.00
RD02	294.50	NN01	772.50	BW02	129.00	AMP03	59.50
RAW02	359.50	Rumelhart02	807.50	BW04	129.00	AMP08	90.00
RAW01	359.50	Rumelhart01	845.50	Cliente02	136.00	AMP11	119.00
GS02	428.50	ED01	957.50	HG01	136.00	AMP12	119.00
Cliente02	431.50	ED04	986.50	GS01	139.00	HG01	133.00
Cliente01	433.50	ED02	1024.00	BW03	169.00	AMP07	135.50

**Tabla 5.21.** Resultados. Clasificación Microsoft. Codificación Abstracta, con valores  $\theta = 2.00$  y  $\Delta = 0.50$ .

ED01		DMA04		Server02		AMP04	
ED01	0.00	DMA04	0.00	Server02	0.00	AMP04	0.00
ED04	21.00	DMA01	177.50	Server03	71.00	AMP05	28.00
ED02	57.00	DMA02	201.00	Server01	73.00	AMP06	28.00
ED03	78.00	DMA03	212.00	Cliente03	121.5	AMP03	38.00
RD02	306.00	ED01	558.50	Cliente02	126.00	AMP09	42.00
RD01	314.00	ED02	573.50	BW02	143.50	HG02	68.00
RAW01	321.50	ED04	574.50	BW04	143.50	AMP12	69.50
RAW02	321.50	ED03	589.50	GS02	144.00	AMP11	69.50
AMP01	361.50	Rumelhart01	608.00	BW01	145.00	AMP07	69.50
BW01	364.50	Rumelhart02	616.50	GS01	150.50	AMP02	69.50

**Tabla 5.22.** Resultados. Segunda Clasificación. Codificación Abstracta, con valores  $\theta = 2.00$ ,  $\Delta = 0.50$ .

ED01		DMA04		Server02		AMP04	
ED01	0.00	DMA04	0.00	Server02	0.00	AMP04	0.00
ED02	37.50	DMA01	208.00	Server03	71.50	AMP06	24.00
ED04	40.00	DMA02	244.50	Server01	78.00	AMP05	24.00
ED03	77.50	DMA03	294.00	Cliente03	93.50	AMP09	62.00
RD02	184.50	NN01	735.50	GS02	106.50	AMP03	77.00
RD01	278.50	Rumelhart01	750.50	GS01	106.50	BW02	84.50
GS01	307.50	Rumelhart02	770.50	BW01	116.50	AMP08	86.00
RAW01	312.00	RD01	1070.50	HG01	118.00	BW04	91.50
RAW02	312.00	ED02	1080.00	BW04	123.50	HG02	113.50
AMP01	312.00	ED03	1105.00	BW02	123.50	HG01	114.00
...	...	...	...	...	...	...	...

**Tabla 5.23.** Resultados. Tercera Clasificación. Codificación Abstracta , con valores  $\theta = 2.00$  y  $\Delta = 0.50$ .

### Codificación en Detalle

A continuación, en las Tablas 5.24, 5.25 y 5.26, se presentan las comparaciones de los resultados a través de las diferentes clasificaciones, para la codificación en detalle de los códigos de referencia *ED01*, *DMA04*, *Server02* y *AMP04*.

En los resultados mostrados en las Tablas 5.24, 5.25 y 5.26, se preserva la relación de semejanza de los códigos similares con respecto al código de referencia, no obstante, existen variaciones en la presentación de los códigos similares al código de referencia. El código *Server02*, muestra en la Tabla 5.24 al código *Cliente03*, como primer código similar, mientras que en la Tabla 5.25 aparece en segundo lugar, además el código *Server03* aparece en los primeros lugares como código similar a *Server02*, a través de las diferentes clasificaciones.

En las Tablas 5.27, 5.28 y 5.29, se muestran los resultados para la codificación en detalle, con valores  $\theta = 2.00$ ,  $\Delta = 0.50$ ,  $\sigma = 0.0125$ , constantes = 1.8, variables = 1.4, para los códigos *ED01*, *DMA04*, *Server02* y *AMP04*. De forma similar a las explicaciones anteriores, se presentan los códigos similares a cada código de referencia, sin embargo, el orden de presentación varía entre cada clasificación.

ED01		DMA04		Server02		AMP04	
ED01	0.00	DMA04	0.00	Server02	0.00	AMP04	0.00
ED04	10.61	DMA01	201.54	Ciente03	81.93	AMP06	19.98
ED02	20.43	DMA02	212.38	Server03	92.39	AMP05	19.98
ED03	31.03	DMA03	226.81	RAW02	103.83	AMP03	41.58
RD02	152.39	Rumelhart02	571.02	GS02	105.84	AMP09	49.38
Server01	159.99	Rumelhart01	574.03	Ciente02	116.26	BW02	50.03
Ciente01	165.37	NN01	585.79	RAW01	124.93	AMP08	51.05
Ciente02	171.09	ED01	707.25	Server01	125.02	AMP01	56.25
Server03	181.56	ED04	716.40	RD02	125.03	AMP11	60.58
RD01	188.75	ED02	725.90	GS01	131.38	AMP12	60.58

**Tabla 5.24.** Resultados. Clasificación Microsoft. Codificación en Detalle, con valores  $\theta = 1.00$ ,  $\Delta = 0.05$ ,  $\sigma = 0.00625$ , constantes = 0.7, variables = 0.9

ED01		DMA04		Server02		AMP04	
ED01	0.00	DMA04	0.00	Server02	0.00	AMP04	0.00
ED02	8.03	DMA01	187.53	Server03	91.29	AMP06	16.18
ED04	11.31	DMA02	199.16	Ciente03	94.93	AMP05	16.18
ED03	19.33	DMA03	202.48	Server01	108.53	AMP03	33.05
RD02	126.26	Rumelhart02	512.83	RAW02	111.83	BW02	44.73
RD01	162.67	Rumelhart01	517.05	GS02	128.03	AMP09	47.38
Ciente01	167.29	NN01	587.23	Ciente02	130.04	AMP08	49.03
Server01	178.76	ED01	668.98	RAW01	130.62	BW03	53.23
Ciente02	185.44	ED02	676.81	GS01	134.82	BW04	56.59
GS02	197.84	ED04	678.78	BW01	154.86	AMP01	61.30

**Tabla 5.25.** Resultados. Segunda Clasificación. Codificación en Detalle, con valores  $\theta = 1.00$ ,  $\Delta = 0.05$ ,  $\sigma = 0.00625$ , constantes = 0.7, variables = 0.9

ED01		DMA04		Server02		AMP04	
ED01	0.0	DMA04	0.00	Server02	0.00	AMP04	0.00
ED02	12.73	DMA03	195.55	Server03	72.69	AMP05	39.40
ED04	13.31	DMA01	196.13	Ciente03	80.93	AMP06	39.40
ED03	26.03	DMA02	209.76	GS02	89.37	AMP03	48.05
RD02	136.22	Rumelhart02	514.81	RD02	97.67	AMP09	49.50
Server01	140.24	Rumelhart01	521.44	Ciente02	103.46	AMP01	55.68
Ciente02	150.31	ED01	584.97	Server01	103.88	BW04	61.14
Server03	155.65	ED02	596.79	GS01	107.27	BW02	63.16
Ciente01	157.93	ED04	597.87	RAW02	110.93	HG03	67.58
RD01	166.28	Ciente01	599.60	RAW01	115.97	AMP08	68.90

**Tabla 5.26.** Resultados. Segunda Clasificación. Codificación en Detalle, con valores  $\theta = 1.00$ ,  $\Delta = 0.05$ ,  $\sigma = 0.00625$ , constantes = 0.7, variables = 0.9



ED01		DMA04		Server02		AMP04	
ED01	0.00	DMA04	0.00	Server02	0.00	AMP04	0.00
ED04	34.81	DMA01	651.49	Ciente03	228.25	AMP05	63.55
ED02	64.45	DMA02	686.95	Server03	255.30	AMP06	63.55
ED03	99.26	DMA03	712.81	RAW02	268.45	AMP03	133.15
Server01	466.33	NN01	1782.64	GS02	276.49	AMP09	145.95
Ciente01	472.14	Rumelhart02	1822.26	Ciente02	317.73	BW02	158.24
RD02	509.19	Rumelhart01	1827.80	RAW01	328.25	AMP08	162.05
Ciente02	509.38	ED01	2292.04	RD02	339.26	AMP01	174.10
Server03	555.86	ED04	2323.83	GS01	372.10	HG03	186.50
RD01	616.43	ED02	2351.79	Server01	380.04	HG02	191.56

**Tabla 5.27.** Resultados. Clasificación Microsoft. Codificación en Detalle, con valores  $\theta = 2.00$ ,  $\Delta = 0.50$ ,  $\sigma = 0.0125$ , constantes = 1.8, variables = 1.4

ED01		DMA04		Server02		AMP04	
ED01	0.00	DMA04	0.00	Server02	0.00	AMP04	0.00
ED02	20.45	DMA01	542.19	Server03	266.99	AMP06	57.55
ED04	33.81	DMA03	570.77	Ciente03	279.85	AMP05	57.55
ED03	54.26	DMA02	576.73	RAW02	314.06	AMP03	92.50
RD02	383.91	Rumelhart02	1486.18	Server01	320.26	AMP09	129.95
RD01	471.59	Rumelhart01	1497.71	RAW01	375.84	BW02	137.86
Ciente01	508.58	NN01	1680.45	GS02	378.21	AMP08	148.45
Server01	554.65	ED01	1932.93	Ciente02	381.69	BW03	157.25
Ciente02	562.76	ED02	1950.61	GS01	398.04	BW04	169.58
GS02	608.01	ED04	1964.51	RD02	453.06	AMP01	176.60

**Tabla 5.28.** Resultados. Segunda Clasificación. Codificación en Detalle, con valores  $\theta = 2.00$ ,  $\Delta = 0.50$ ,  $\sigma = 0.0125$ , constantes = 1.8, variables = 1.4

ED01		DMA04		Server02		AMP04	
ED01	0.00	DMA04	0.00	Server02	0.00	AMP04	0.00
ED02	35.45	DMA01	579.79	Server03	208.99	AMP05	120.40
ED04	37.81	DMA03	582.80	Ciente03	235.85	AMP06	120.40
ED03	73.26	DMA02	618.30	GS02	245.94	AMP09	144.60
Server01	383.28	Rumelhart02	1498.13	RD02	283.34	AMP03	146.10
RD02	393.64	Rumelhart01	1499.80	Ciente02	298.35	AMP01	167.75
Ciente02	417.03	Ciente01	1748.41	GS01	300.94	BW04	183.48
Ciente01	432.59	ED01	1753.14	Server01	305.76	BW02	192.31
Server03	442.10	ED02	1787.45	RAW02	320.34	HG02	207.43
Server02	479.15	ED04	1790.38	RAW01	331.44	HG03	210.35

**Tabla 5.29.** Resultados. Tercera Clasificación. Codificación en Detalle, con valores  $\theta = 2.00$ ,  $\Delta = 0.50$ ,  $\sigma = 0.0125$ , constantes = 1.8, variables = 1.4

## Identificación de subsecuencias en Códigos Fuente

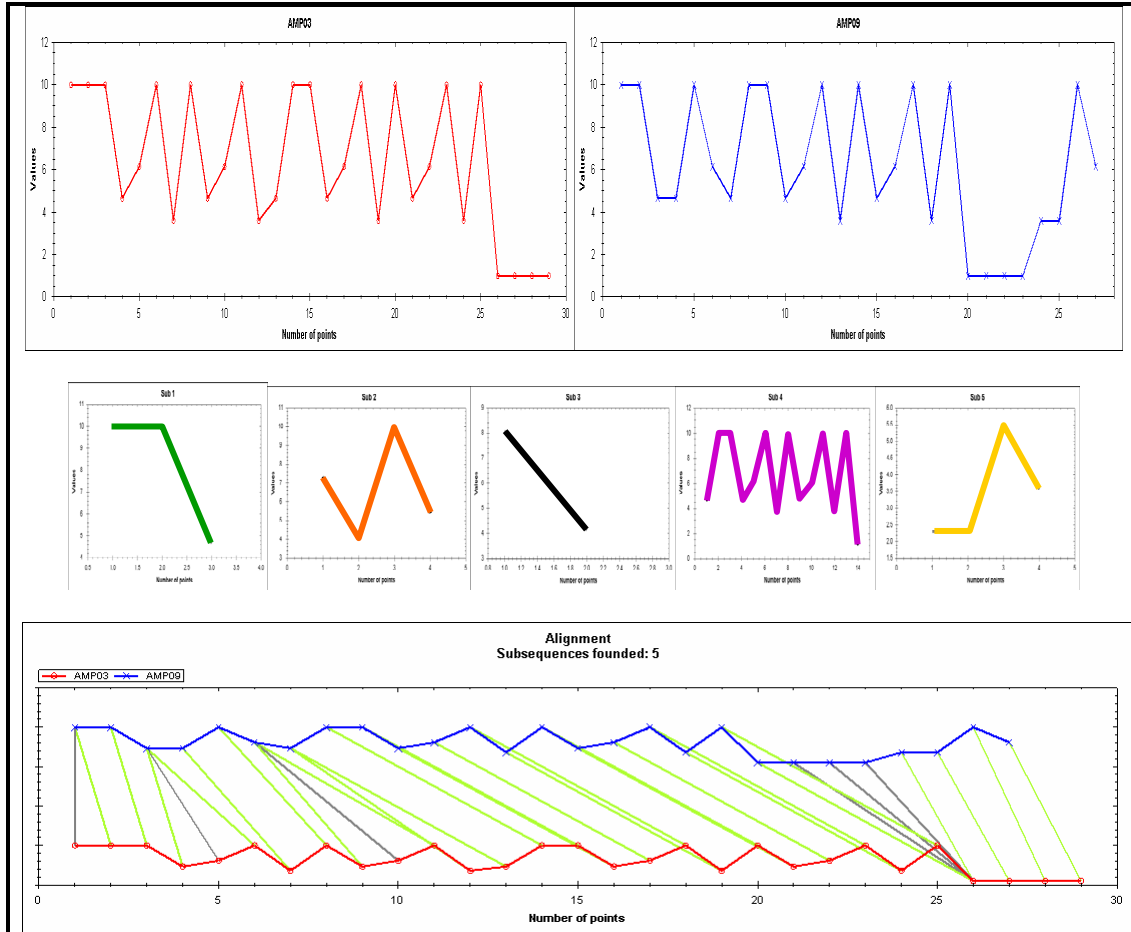
En esta sección, se muestran los resultados de la aplicación del algoritmo para identificar subsecuencias propuesto en el Capítulo 3, en las transformaciones realizadas a los códigos fuente, mediante la codificación abstracta y en detalle. Algunos de estos resultados pueden encontrarse en [Mirón-Bernal07b].

Una vez que se ha representado el código fuente en una secuencia numérica, se aplica la técnica *FDTW* para obtener la matriz de distancias entre dos códigos. Con esta matriz se determina el *warp path* sobre el cual se identifican las subsecuencias.

En la transformación propuesta en el Capítulo 3, se presenta la idea que el código fuente, posee una estructura que caracteriza al código fuente, en base a las instrucciones que intervienen en su sintaxis. Las subsecuencias detectadas con el algoritmo propuesto, serán parte de esta estructura. A continuación se muestran los resultados obtenidos, utilizando la codificación abstracta con valores  $\theta=1.00$  y  $\Delta=0.05$ . Tomando como referencia los códigos *AMP03* y *AMP09*, mostrados en las Figura 5.32 y 5.33 respectivamente., se identifican cinco subsecuencias similares a estos códigos, Ver Figura 5.31. El resumen comparativo se muestra en la Tabla 5.30.

Resumen ( AMP03 , AMP09 )	
<i>Distancia FDTW :</i>	33.65
<i>Subsecuencias Detectadas :</i>	5
<i>Valor de Similitud :</i>	<b>0.9322</b>

**Tabla 5.30.** Resumen comparativo entre los códigos *AMP03* y *AMP09*.



**Figura 5.31.** Subsecuencias identificadas en los códigos fuente. *AMP03* y *AMP09*.

En la Figura 5.31, se muestra en color rojo el código *AMP03*, en color azul el código *AMP09*. Las subsecuencias identificadas similares a ambos códigos, se muestran gráficamente. En la parte inferior de la Figura 5.31 se muestra el alineamiento realizado por *FDTW*. En las Figura 5.32 y 5.33, se muestra la representación en código de las subsecuencias identificadas.

```

private void Amplificar03()
{
    int x, y, i, j;
    Color p;

    for ( x = 0; i < input.Width; i++)
    {
        for ( y = 0; j < input.Height; j++)
        {
            p = input.GetPixel(x,y);
            for (i = 0; i < factor;i++)
            {
                for (j = 0; j < factor; j++)
                {
                    output.SetPixel( x * factor + i,
                                     y * factor + j, p);
                }
            }
        }
    }
}

```

Figura 5.32. Código *AMP03*. Subsecuencias identificadas con *AMP09*.

```

private void Amplificar09()
{
    int x, y, i, j;
    x = y = 0;
    Color p;
    do{
        while ( y < input.Height )
        {
            p = input.GetPixel(x, y);
            for (int x = 0; x < factor; x++)
            {
                for (int y = 0; y < factor; y++)
                {
                    output.SetPixel( i * factor + x
                                     , j * factor + y, p);
                }
            }
            y++;
        }
        x++;
    } while ( x < input.Width );
}

```

Figura 5.33. Código *AMP09*. Subsecuencias identificadas con *AMP03*.

En las Figuras 5.32 y 5.33, se muestran en recuadros de color, las instrucciones correspondientes a cada subsecuencia identificada. La subsecuencia número 1, de la Figura 5.31, se corresponde en recuadros color verde; la subsecuencia número 2, en recuadros color naranja; la subsecuencia número 3, en recuadros color negro; la subsecuencia número 4 en recuadros color morado; y la subsecuencia número 5, en recuadros color amarillo.

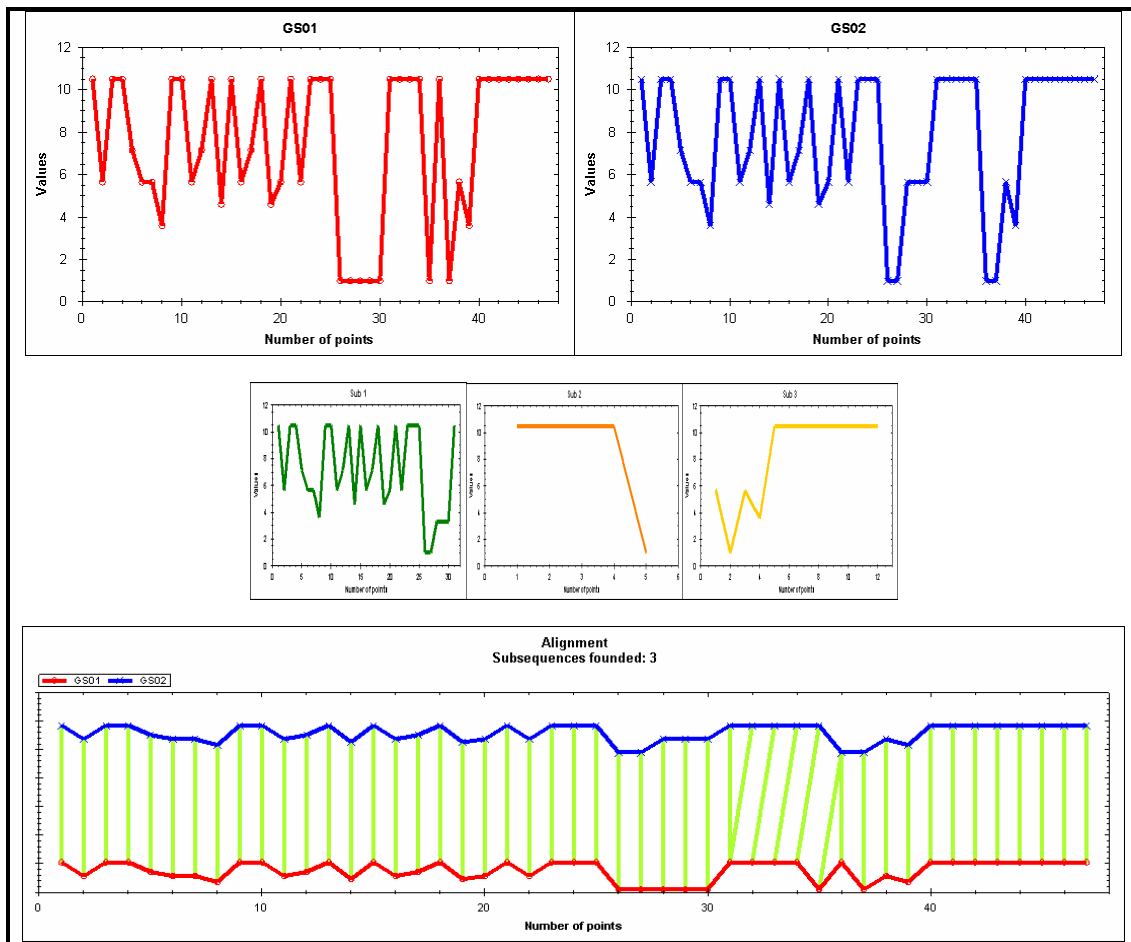
De esta forma, se identifican subsecuencias similares entre códigos fuentes utilizando el algoritmo propuesto en el Capítulo 3; el cuál, no requiere realizar el proceso de extracción de características. Se comparan los códigos fuente representados en secuencias numéricas, y con el algoritmo se detectan las subsecuencias similares sin supervisión alguna.

Ahora se presentan las subsecuencias identificadas entre los códigos GS01 y GS02. En la Tabla 5.31, se muestra le resumen comparativo entre los códigos mencionados.

El código GS01, convierte una imagen en colores a una imagen en escala de grises, utilizando una fórmula de conversión ponderada, mientras que el código GS02 utiliza una fórmula de conversión promediada entre los planos R,G,B, que componen el color de un píxel de la imagen.

<b>Resumen ( GS01 , GS02 )</b>	
<i>Distancia FDTW :</i>	<i>23.45</i>
<i>Subsecuencias Detectadas :</i>	<b>3</b>
<i>Valor de Similitud :</i>	<b>0.9493</b>

**Tabla 5.31.** Resumen comparativo entre los códigos *GS01* y *GS02*



**Figura 5.34.** Subsecuencias identificadas en los códigos fuente. *GS01* y *GS02*.

En las Figuras 5.35 y 5.36, se muestran en recuadros de color, las instrucciones correspondientes a cada subsecuencia identificada. La subsecuencia número 1, de la Figura 5.34, se corresponde en recuadros color verde; la subsecuencia número 2, en recuadros color naranja; la subsecuencia número 3, en recuadros color amarillo.

```
private void ponderadoToolStripMenuItem_Click
(object sender, EventArgs e){
try
{NewImage temp =(NewImage)ActiveMdiChild;
if (temp != null)
{Bitmap output, img = temp.input;
output = new Bitmap (img.Width,
img.Height);
for (int i = 0; i < img.Size.Width; i++)
{for (int j = 0; j < img.Size.Height; j++)
{Color c = img.GetPixel(i, j);
int lum = (int) (c.R * 0.3 +
c.G * 0.59 + c.B * 0.11);

output.SetPixel(i, j, Color .FromArgb(
lum, lum, lum));
}
SetProgress((i/img.Width)*100);
}
NewImage salida = new NewImage (output);

salida.SetParent (this);
salida.ChangeName ("GrayPond", temp.name);
salida.Show();
SetProgress(100);
}
} catch (Exception ex) {SetProgress(100);}
}
```

Figura 5.35. Código *GS01*. Subsecuencias identificadas con *GS02*.

```

private void promedioToolStripMenuItem_Click
(object sender, EventArgs e){
try
{NewImage temp =(NewImage)ActiveMdiChild;
if (temp != null)
{Bitmap output, img = temp.input;
output = new Bitmap (img.Width,
img.Height);
for (int i = 0; i < img.Size.Width; i++)
{for (int j = 0; j < img.Size.Height; j++)
{Color c = img.GetPixel(i, j);
double value = (double)(c.R + c.G + c.B);
value /= 768; //256*3
value *= 256;
int lum = Convert.ToInt32(value);

output.SetPixel(i, j, Color.FromArgb(
lum, lum, lum));
}
SetProgress((i / img.Width) * 100);

}NewImage salida = new NewImage(output);
salida.SetParent( this );
salida.ChangeName("GrayProm", temp.name);
salida.Show();
SetProgress(100);
}
}catch (Exception ex){SetProgress(100);}
}

```

**Figura 5.36.** Código *GS02*. Subsecuencias identificadas con *GS01*.

En los anexos, se muestran las tablas de distancias *FDTW*, para el conjunto de datos. Las tablas de distancias con las diferentes clasificaciones y valores de codificación, se encuentran en el CD correspondiente a esta tesis.



## Referencias del capítulo

[Mirón-Bernal07d] M. Mirón-Bernal, H. Coyote-Estrada, J. Figueroa-Nazuno. “*Identificación de subsecuencias similares en lenguajes de programación de Alto Nivel*”, Decimoséptima reunión de otoño de comunicaciones, computación, electrónica y exposición industrial. IEEE – ROC&C2007. Acapulco, Guerrero, México. 2007.

[Mirón-Bernal07a] M. Mirón-Bernal, H. Coyote-Estrada, J. Figueroa-Nazuno. “*Cómputo de la Semejanza entre códigos fuente*” Reporte Técnico. Centro de Investigación en Computación. Instituto Politécnico Nacional. U.P. “Adolfo López Mateos” .México, DF. 2007.

[Mirón-Bernal07b] M. Mirón-Bernal, H. Coyote-Estrada, J. Figueroa-Nazuno. “*Code Similarity on High Level Programs*”, Cornell University Library. e-Print Archive. arXiv:0710.5547v1 [cs.CV] 2007.

[Mirón-Bernal06] M. Mirón-Bernal, A. Angeles-Yreta, J. Figueroa-Nazuno, “*Clasificación de Series de Tiempo mediante la identificación de subsecuencias similares*”, Memorias del XLIXI Congreso Nacional de Física. S.L.P, S.L.P. ISSN 0187-4713. Octubre 2006.

[Mirón-Bernal05] M. Mirón-Bernal, A. Angeles-Yreta, J. Figueroa-Nazuno, “*Complejidad y Dependencias de Rango Extendido en Series de Tiempo*”, Memorias del XLVIII Congreso Nacional de Física. Guadalajara, Guadalajara. ISSN 0187-4713. Octubre 2005.

# **CAPÍTULO 6**

---

## **CONCLUSIONES**

Se presentó una nueva aproximación al problema del cómputo de similitud entre códigos fuentes, donde **no** se involucra el proceso de extracción de características del paradigma clásico de similitud; los resultados sugieren que no siempre debe incluirse este proceso.

Se presentaron dos nuevas formas de representación de códigos fuente de un lenguaje de programación de Alto Nivel. Ambas formas de representación incluyen la transformación del código fuente en una *secuencia numérica*.

- i. La primera transformación *codificación abstracta*, se basa en la clasificación de operadores del lenguaje de programación; las reglas de codificación, involucran la representación de un conjunto de operadores como parte de una categoría a la cual se asigna un valor de identificación único.
- ii. La segunda transformación *codificación en detalle*, se basa en la idea de identificar de manera única, cada operador que interviene en la clasificación de operadores de un lenguaje de programación; las reglas de codificación, incluyen la identificación de el conjunto de palabras reservadas, constantes y variables.

Los resultados obtenidos en este trabajo, al realizar la variación de parámetros y clasificación de operadores, sugieren que, la estructura del código fuente a representar en secuencia numérica, se preserva. Los elementos similares a un código fuente de referencia, pueden variar en el orden de presentación (primer, segundo, tercer código similar...); sin embargo, conservarán el grado de semejanza que existe entre las estructuras de los códigos a comparar.

Se presentó un nuevo algoritmo no supervisado y automático para la identificación de *subsecuencias* comunes entre secuencias numéricas. El algoritmo propuesto, no incluye el proceso de extracción/selección de características y es generalizable a aquellos fenómenos que puedan ser representados en una secuencia numérica.

La búsqueda de diferentes, y nuevas representaciones de las entidades a comparar, permite el uso de técnicas y métodos de diversas áreas de especialización para el cómputo de similitud.

En el problema del análisis y cómputo de similitud, no existe un criterio de comparación universal; además, no es posible realizar la medición del *error* de comparación.

# PUBLICACIONES

---

- I. [Mirón-Bernal07d] M. Mirón-Bernal, H. Coyote-Estrada, J. Figueroa-Nazuno. “Identificación de subsecuencias similares en lenguajes de programación de Alto Nivel”, Decimoséptima reunión de otoño de comunicaciones, computación, electrónica y exposición industrial. IEEE – ROC&C2007. Acapulco, Guerrero, México. 2007.
- II. [Mirón-Bernal07c] M. Mirón-Bernal, F. Colorado-Rodríguez, “*Elaboración y Distribución de Objetos de Aprendizaje utilizando el modelo SCORM*”, Congreso Metodologías 2007, México, D.F. ISBN: 978-970-36-0431-9. Noviembre 2007.
- III. [Mirón-Bernal07b] M. Mirón-Bernal, H. Coyote-Estrada, J. Figueroa-Nazuno. “*Code Similarity on High Level Programs*”, Cornell University Library. e-Print Archive. arXiv:0710.5547v1 [cs.CV] 2007.
- IV. [Mirón-Bernal07a] M. Mirón-Bernal, H. Coyote-Estrada, J. Figueroa-Nazuno. “*Cómputo de la Semejanza entre códigos fuente*” Reporte Técnico. Centro de Investigación en Computación. Instituto Politécnico Nacional. U.P. “Adolfo López Mateos” .México, DF. 2007.

- V. [Figuroa-Nazuno06] J. Figuroa-Nazuno, A. Angeles-Yreta, J. Medina-Apodaca, V. Ortega-González, K. Ramírez-Amaro, M. Mirón-Bernal, V. Landassuri-Moreno. *“Sobre el problema de Semejanza”*. Reporte Técnico. Centro de Investigación en Computación. Instituto Politécnico Nacional. Unidad Profesional “Adolfo López Mateos” .ISBN: 970-36-0343-2. México, DF. 2006.
- VI. [Mirón-Bernal06] M. Mirón-Bernal, A. Angeles-Yreta, J. Figuroa-Nazuno, *“Clasificación de Series de Tiempo mediante la identificación de subsecuencias similares”*, Memorias del XLIXI Congreso Nacional de Física. S.L.P, S.L.P. ISSN 0187-4713. Octubre 2006.
- VII. [Mirón-Bernal05] M. Mirón-Bernal, A. Angeles-Yreta, J. Figuroa-Nazuno, *“Complejidad y Dependencias de Rango Extendido en Series de Tiempo”*, Memorias del XLVIII Congreso Nacional de Física. Guadalajara, Guadalajara. ISSN 0187-4713. Octubre 2005.

# REFERENCIAS

---

[Angeles05e] Angeles-Yreta A., Figueroa-Nazuno J., Ramírez-Amaro K. “*Búsqueda de Semejanza entre Objetos 3D por Indexado*”. Decimosexta Reunión de Otoño Comunicaciones, Computación, Electrónica y Exposición Industrial, ROC&C’ 2005.

[Angeles06] A. Angeles-Yreta. “*Cómputo de la Similitud entre Figuras Geométricas*”. Tesis de Maestría en Ciencias de la Computación del Centro de Investigación en Computación del Instituto Politécnico Nacional, 2006.

[Arwin06] Christian Arwin, S. M. M. Tahaghoghi. “*Plagiarism detection across programming languages*”. Proceedings of the 29th Australasian Computer Science Conference – Volume. pp. 277 - 286 , 2006 .

[Baker98] Brenda S. Baker,Raffaele Giancarlo. “*Longest Common Subsequence from Fragments via Sparse Dynamic Programming*”. Proceedings of the 6th Annual European Symposium on Algorithms. pp. 79 – 90, 1998.

[Basit05] Hamid Abdul Basit, Stan Jarzabek. “*Detecting higher-level similarity patterns in programs*”, Proceedings of the 10th European software engineering conference. ACM SIGSOFT International Symposium on Foundations of software Engineering, pp. 156 - 165 , 2005

---

[Berndt96] Berndt D. J., Clifford J. “*Finding Patterns in Time Series: A Dynamic Programming Approach*”. Advances in Knowledge Discovery and Data Mining, AAAI/MIT, pp. 229-248, 1996.

[Chuanjun05] Chuanjun Li, B. Prabhakaran. “*A similarity measure for motion stream segmentation and recognition*”. Proceedings of the 6th international workshop on Multimedia data mining: mining integrated media and complex data. Pp. 89 – 94. 2005.

[Engels07] Steve Engels, Vivek Lakshmanan, Michelle Craig. “*Plagiarism detection using feature-based neural networks*”. Proceedings of the 38th SIGCSE technical symposium on Computer science. pp. 34 – 38, 2007.

[Figuroa-Nazuno06] J. Figuroa-Nazuno, A. Angeles-Yreta, J. Medina-Apodaca, V. Ortega-González, K. Ramírez-Amaro, M. Mirón-Bernal, V. Landassuri-Moreno. “*Sobre el problema de Semejanza*”. Reporte Técnico. Centro de Investigación en Computación. Instituto Politécnico Nacional. Unidad Profesional “Adolfo López Mateos”. ISBN: 970-36-0343-2. México, DF. 2006.

[Hao05] Dan Hao ,Ying Pan, Lu Zhang, Wei Zhao, Hong Mei, Jiasu Sun. “*A similarity-aware approach to testing based fault localization*”. Proceedings of the 20th IEEE/ACM International Conference on Automated software engineering. pp. 291 – 294, 2005.

[Hirschberg75] D. S. Hirschberg. “*A linear space algorithm for computing maximal common subsequences*”. Communications of the ACM Volume 18, Issue 6 (June 1975), pp. 341 – 343, 1975.

[Hirschberg77] D. S. Hirschberg . “*Algorithms for the Longest Common Subsequence Problem*” . Journal of the ACM , Volume 24 , Issue 4 , pp. 664 – 675, 1977.



---

[Hunt77] James W. Hunt, Thomas G. Szymanski. “*A fast algorithm for computing longest common subsequences*”. Communications of the ACM Volume 20, Issue 5, pp. 350 – 353, 1977 .

[Jang00] Jang J.S.R, Gao M.Y. “*A Query-by-Singing System based on Dynamic Programming*”. International Workshop on Systems Resolutions (the 8<sup>th</sup> Bellman Continuum), pp. 85-89, 2000.

[Kamejima97] Kamejima, K. “*Parallel distributed scheme for stochastic coding of self-similar patterns*”, Parallel Algorithms/Architecture Synthesis, 1997. Proceedings of the Second Aizu International Symposium, pp.192 – 199. 1997.

[Karagiannis04] T. Karagiannis, M.Molle, M.Faloutsos, “*Long Range Dependence, ten years of Internet Traffic Modeling*”, IEEE Internet Computing, September-October 2004.

[Keogh02] Keogh E. “*Exact Indexing of Dynamic Time Warping*”. In Proceedings of the 28<sup>th</sup> VLDB Conference, pp. 406-417, 2002.

[Kim01] Kim S. W., Park S., Chu W. W. “*An Index-Based Approach for Similarity Search Supporting Time Warping in Large Sequence Database*”. In Proceedings of the 17<sup>th</sup> International Conference on Data Engineering (ICDE’01), pp. 607-614, 2001.

[Kim02] Jong-Wook Kim; Sang Woo Kim; PooGyeon Park; Tae Joon Park. “*On the similarities between binary-coded GA and real-coded GA inside search space*”. Proceedings of the Evolutionary Computation Congress . Pags:681 – 686. 2002.

---

[Komondoor01] Raghavan Komondoor, Susan Horwitz. “*Tool Demonstration: Finding Duplicated Code Using Program Dependences*”. Proceedings of the 10th European Symposium on Programming Languages and Systems pp. 383 – 386, 2001.

[Kontogiannis93] Kostas Kontogiannis. “*Program representation and behavioural matching for localizing similar code fragments*”. Proceedings of the 1993 conference of the Center for Advanced Studies on Collaborative research: software engineering. Volume 1, Pages 194 – 205, 1993.

[Krinke01] Krinke, J. “*Identifying similar code with program dependence graphs*”. Proceedings of the Eighth Working Conference on Reverse Engineering. pp. 301 – 309. 2001.

[Kubo05] Kubo, A.; Washizaki, H.; Takasu, A.; Fukazawa, Y. “*Analyzing relations among software patterns based on document similarity*”. Proceedings of the Information Technology: Coding and Computing. ITCC 2005. International Conference Page(s): 298 - 303 . 2005

[Lynch04] Lynch, P.; Luan, X.; Prettyman, M.; Mericle, L.; Borkmann, E.; Schlaifer, J. “*An evaluation of new and old similarity ranking algorithms*”. Proceedings of the Information Technology: Coding and Computing. ITCC 2004. Page(s): 148 - 149 Vol.2. 2004

[Maier78] David Maier. “*The Complexity of Some Problems on Subsequences and Supersequences*”. Journal of the ACM, Volume 25 , Issue 2 . pp. 322 – 336, 1978.

[Maletic00] Jonathan I. Maletic, Andrian Marcus. “*Using latent semantic analysis to identify similarities in source code to support program understanding*”. Proceedings of the 12th IEEE International Conference on Tools with Artificial Intelligence. Pp. 46 , 2000.

[Mann06] Samuel Mann, Zeldia Frew. “*Similarity and originality in code: plagiarism and normal variation in student assignments*”. Proceedings of the 8th Australian conference on Computing education. pp. 143 - 150, 2006.

[Matsumoto98] M. Matsumoto & T. Nishimura, “*Mersenne twister: a 623-dimensionally equidistributed uniform pseudorandom number generator*”, ACM Trans. Model. Comput. Simul. 8, 3 (1998).

[Mishne04]G. Mishne and M. de Rijke. “*Source Code Retrieval using Conceptual Similarity*”. Proceedings of the *Recherche d’Information Assiteé par Ordinateur*. 2004.

[Mirón-Bernal05] M. Mirón-Bernal, A. Angeles-Yreta, J. Figueroa-Nazuno, “*Complejidad y Dependencias de Rango Extendido en Series de Tiempo*”, Memorias del XLVIII Congreso Nacional de Física. Guadalajara, Guadalajara. ISSN 0187-4713. Octubre 2005.

[Mirón-Bernal06] M. Mirón-Bernal, A. Angeles-Yreta, J. Figueroa-Nazuno, “*Clasificación de Series de Tiempo mediante la identificación de subsecuencias similares*”, Memorias del XLIXI Congreso Nacional de Física. S.L.P, S.L.P. ISSN 0187-4713. Octubre 2006.

[Mirón-Bernal07a] M. Mirón-Bernal, H. Coyote-Estrada, J. Figueroa-Nazuno. “*Cómputo de la Semejanza entre códigos fuente*” Reporte Técnico. Centro de Investigación en Computación. Instituto Politécnico Nacional. U.P. “Adolfo López Mateos” .México, DF. 2007.

[Mirón-Bernal07b] M. Mirón-Bernal, H. Coyote-Estrada, J. Figueroa-Nazuno. “*Code Similarity on High Level Programs*”, Cornell University Library. e-Print Archive. arXiv:0710.5547v1 [cs.CV] 2007.

[Mirón-Bernal07d] M. Mirón-Bernal, H. Coyote-Estrada, J. Figueroa-Nazuno. “*Identificación de subsecuencias similares en lenguajes de programación de Alto Nivel*”, Decimoséptima reunión de otoño de comunicaciones, computación, electrónica y exposición industrial. IEEE – ROC&C2007. Acapulco, Guerrero, México. 2007.

[MSDN05] The Visual Studio Combined Help Collection. Microsoft Visual Studio 2005 Documentation. Microsoft Corporation.

[Myles05] Ginger Myles, Christian Collberg. “*K-gram based software birthmarks*” Proceedings of the 2005 ACM symposium on Applied computing. pp. 314 - 318, 2005.

[Ohno06] Asako Ohno, Hajime Murao. “*Measuring Source Code Similarity Using Reference Vectors*”. First International Conference on Innovative Computing, Information and Control - Volume II (ICICIC'06) pp. 92-95, 2006.

[Ramírez04]K. Ramírez-Amaro, H. Jiménez-Hernández, J. Figueroa-Nazuno, “*Análisis de los Índices de Teleconexión Atmosféricos Empleando Técnicas de Mapas Recurrentes Y Modelos Casuales*”; IEEE Reunión de Otoño de Comunicaciones y Computación, Noviembre 2004.

[Sager06] Tobias Sager, Abraham Bernstein, Martin Pinzger, Christoph Kiefer. “*Detecting similar Java classes using tree algorithms*”. Proceedings of the 2006 international workshop on Mining software repositories. pp 65 – 71, 2006.

---

[Schleimer03] Saul Schleimer, Daniel S. Wilkerson, Alex Aiken. “*Winnowing: local algorithms for document fingerprint*”. International Conference on Management of Data. Proceedings of the 2003 ACM SIGMOD international conference on Management of Data. San Diego, California, pp. 76 - 85 , 2003 .

[Silverman&Morgan90] Silverman, H. F. & Morgan, “*The application of dynamic programming to connected speech recognition*”. IEEE ASSP Magazine, 7--24. D. P. July, 1990.

[Sun04] Fu-Shing Sun; Chun-Hung Tzeng. “*A mathematical model of similarity and clustering*”. Proceedings of the Information Technology: Coding and Computing. ITCC 2004. Page(s): 460 – 464, 2004.

[Tung06] Anthony K. H. Tung, Rui Zhang, Nick Koudasx Beng Chin Ooi. “*Similarity search: a matching based approach*”. Proceedings of the 32nd international conference on Very large data. pp. 631 - 642, 2006.

[Tversky77] Tversky A., Shafir E. “*Preference, Belief, and Similarity*”. Selected Writings, Ed. MIT Press. Paperback: 1039 pp., ISBN 026270093X. Cambridge, MA. 2004.

[Ullman76] J. D. Ullman ,A. V. Aho, D. S. Hirschberg. “*Bounds on the Complexity of the Longest Common Subsequence Problem*”. Journal of the ACM. Volume 23 , Issue 1. pp. 1 – 12, 1976.

[Xie06] Xinrong Xie, Denys Poshyvanyk, Andrian Marcus. “*3D visualization for concept location in source code*”. Proceeding of the 28th international conference on Software engineering. pp.839 – 842, 2006.

[Zeinalipour-Yazti06] Demetrios Zeinalipour-Yazti, Song Lin, Dimitrios Gunopulos. “*Distributed spatio-temporal similarity search*”. Proceedings of the 15th ACM international conference on Information and knowledge management, pp 14 – 23, 2006.

# **ANEXOS**

---

**ANEXO A.** Tabla de Resultados, utilizando el valor *Sim.* Clasificación Microsoft.

Parámetros  $\theta = 1.00$  y  $\Delta = 0.05$ .

1ª Parte	AMP01	AMP02	AMP03	AMP04	AMP05	AMP06	AMP07	AMP08	AMP09
AMP01	1.0000	0.9196	0.8993	0.8899	0.8666	0.8666	0.9196	0.8622	0.8870
AMP02	0.9172	1.0000	0.9443	0.9159	0.8775	0.8775	1.0000	0.8375	0.8761
AMP03	0.8920	0.9420	1.0000	0.9653	0.9245	0.9245	0.9420	0.8856	0.9260
AMP04	0.8815	0.9121	0.9651	1.0000	0.9715	0.9715	0.9121	0.9476	0.9759
AMP05	0.8492	0.8656	0.9204	0.9701	1.0000	1.0000	0.8656	0.9768	0.9593
AMP06	0.8492	0.8656	0.9204	0.9701	1.0000	1.0000	0.8656	0.9768	0.9593
AMP07	0.9172	1.0000	0.9443	0.9159	0.8775	0.8775	1.0000	0.8375	0.8761
AMP08	0.8475	0.8254	0.8818	0.9462	0.9773	0.9773	0.8254	1.0000	0.9697
AMP09	0.8797	0.8719	0.9265	0.9761	0.9616	0.9616	0.8719	0.9709	1.0000
AMP10	0.9172	1.0000	0.9443	0.9159	0.8775	0.8775	1.0000	0.8375	0.8761
AMP11	0.9379	0.9799	0.9572	0.9257	0.8868	0.8868	0.9799	0.8471	0.8860
AMP12	0.9379	0.9799	0.9572	0.9257	0.8868	0.8868	0.9799	0.8471	0.8860
BW01	0.9294	0.8335	0.8542	0.8839	0.8622	0.8622	0.8335	0.8932	0.9202
BW02	0.8541	0.8615	0.9082	0.9069	0.9027	0.9027	0.8615	0.9230	0.9185
BW03	0.8340	0.8525	0.9063	0.9161	0.9417	0.9417	0.8525	0.9528	0.9393
BW04	0.8521	0.8615	0.9052	0.9069	0.9027	0.9027	0.8615	0.9201	0.9185
Ciente01	0.8598	0.7679	0.7517	0.7492	0.7444	0.7444	0.7679	0.7722	0.8024
Ciente02	0.8716	0.8101	0.8173	0.8188	0.8078	0.8078	0.8101	0.8435	0.8728
Ciente03	0.8874	0.8189	0.8480	0.8480	0.8492	0.8492	0.8189	0.8924	0.9102
DMA01	0.0280	0.0272	0.0261	0.0260	0.0248	0.0248	0.0272	0.0253	0.0263
DMA02	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
DMA03	0.0245	0.0274	0.0321	0.0096	0.0124	0.0124	0.0274	0.0262	0.0295
DMA04	0.1254	0.1249	0.1329	0.1315	0.1250	0.1250	0.1249	0.1147	0.1197
ED01	0.6946	0.6122	0.6110	0.6416	0.6218	0.6218	0.6122	0.6628	0.6955
ED02	0.7225	0.6230	0.6174	0.6467	0.6271	0.6271	0.6230	0.6758	0.6984
ED03	0.7413	0.6463	0.6398	0.6669	0.6483	0.6483	0.6463	0.6944	0.7209
ED04	0.7179	0.6355	0.6334	0.6638	0.6466	0.6466	0.6355	0.6881	0.7180
GS01	0.8925	0.8125	0.8216	0.8144	0.7903	0.7903	0.8125	0.8715	0.9034
GS02	0.8845	0.8312	0.8533	0.8491	0.8280	0.8280	0.8312	0.9001	0.9385
HG01	0.8901	0.8345	0.8799	0.9132	0.9008	0.9008	0.8345	0.9438	0.9769
HG02	0.8267	0.8345	0.8988	0.9133	0.9254	0.9254	0.8345	0.9462	0.9365
HG03	0.8194	0.8078	0.8748	0.8759	0.9020	0.9020	0.8078	0.9455	0.9228
NN01	0.2703	0.2943	0.3067	0.2946	0.3022	0.3022	0.2943	0.3025	0.2990
RAW01	0.8227	0.8053	0.7968	0.8125	0.7736	0.7736	0.8053	0.7847	0.8087
RAW02	0.8227	0.7371	0.7313	0.7648	0.7281	0.7281	0.7371	0.7614	0.8037
RD01	0.7848	0.7558	0.7151	0.6990	0.6687	0.6687	0.7558	0.6925	0.7208
RD02	0.8160	0.7233	0.7278	0.7516	0.7192	0.7192	0.7233	0.7397	0.7834
Rumelhart01	0.4522	0.4919	0.4825	0.4713	0.4736	0.4736	0.4919	0.5035	0.5120
Rumelhart02	0.4645	0.5044	0.4852	0.4713	0.4638	0.4638	0.5044	0.4936	0.5125
Server01	0.8643	0.8002	0.8129	0.7761	0.7366	0.7366	0.8002	0.8450	0.8710
Server02	0.8596	0.8097	0.8313	0.7932	0.7615	0.7615	0.8097	0.8675	0.8809
Server03	0.8744	0.8201	0.8293	0.7934	0.7617	0.7617	0.8201	0.8618	0.8833



**ANEXO A.** Tabla de Resultados, utilizando el valor *Sim*. Clasificación Microsoft.

Parámetros  $\theta = 1.00$  y  $\Delta = 0.05$ .

2ª Parte	AMP10	AMP11	AMP12	BW01	BW02	BW03	BW04	Ciente01	Ciente02
AMP01	0.9196	0.9409	0.9409	0.9285	0.8613	0.8502	0.8595	0.8570	0.8742
AMP02	1.0000	0.9803	0.9803	0.8263	0.8644	0.8630	0.8644	0.7562	0.8083
AMP03	0.9420	0.9563	0.9563	0.8418	0.9065	0.9094	0.9034	0.7286	0.8081
AMP04	0.9121	0.9239	0.9239	0.8733	0.9047	0.9185	0.9047	0.7246	0.8087
AMP05	0.8656	0.8783	0.8783	0.8422	0.8954	0.9405	0.8954	0.7053	0.7871
AMP06	0.8656	0.8783	0.8783	0.8422	0.8954	0.9405	0.8954	0.7053	0.7871
AMP07	1.0000	0.9803	0.9803	0.8263	0.8644	0.8630	0.8644	0.7562	0.8083
AMP08	0.8254	0.8389	0.8389	0.8803	0.9190	0.9528	0.9159	0.7428	0.8302
AMP09	0.8719	0.8845	0.8845	0.9139	0.9175	0.9417	0.9175	0.7854	0.8673
AMP10	1.0000	0.9803	0.9803	0.8263	0.8644	0.8630	0.8644	0.7562	0.8083
AMP11	0.9799	1.0000	1.0000	0.8297	0.8787	0.8725	0.8787	0.7581	0.8079
AMP12	0.9799	1.0000	1.0000	0.8297	0.8787	0.8725	0.8787	0.7581	0.8079
BW01	0.8335	0.8399	0.8399	1.0000	0.9191	0.8892	0.9191	0.8380	0.9028
BW02	0.8615	0.8786	0.8786	0.9138	1.0000	0.9571	1.0000	0.8034	0.8874
BW03	0.8525	0.8655	0.8655	0.8757	0.9547	1.0000	0.9526	0.7762	0.8584
BW04	0.8615	0.8786	0.8786	0.9138	1.0000	0.9550	1.0000	0.8034	0.8874
Ciente01	0.7679	0.7743	0.7743	0.8391	0.8168	0.8021	0.8168	1.0000	0.9186
Ciente02	0.8101	0.8135	0.8135	0.8996	0.8908	0.8697	0.8908	0.9153	1.0000
Ciente03	0.8189	0.8325	0.8325	0.9064	0.9150	0.9064	0.9150	0.8670	0.9392
DMA01	0.0272	0.0266	0.0266	0.0284	0.0266	0.0253	0.0266	0.0286	0.0274
DMA02	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
DMA03	0.0274	0.0328	0.0328	0.0253	0.0211	0.0253	0.0211	0.0428	0.0470
DMA04	0.1249	0.1355	0.1355	0.1291	0.1149	0.1157	0.1149	0.1432	0.1469
ED01	0.6122	0.6141	0.6141	0.7118	0.6687	0.6523	0.6687	0.7081	0.7233
ED02	0.6230	0.6246	0.6246	0.7417	0.6840	0.6634	0.6840	0.7037	0.7156
ED03	0.6463	0.6474	0.6474	0.7660	0.7068	0.6912	0.7068	0.7248	0.7391
ED04	0.6355	0.6369	0.6369	0.7361	0.6915	0.6800	0.6915	0.7292	0.7468
GS01	0.8125	0.8166	0.8166	0.9434	0.8954	0.8790	0.8954	0.8098	0.8799
GS02	0.8312	0.8349	0.8349	0.9336	0.9309	0.9006	0.9309	0.8097	0.8795
HG01	0.8345	0.8447	0.8447	0.9184	0.9560	0.9345	0.9560	0.8088	0.8747
HG02	0.8345	0.8478	0.8478	0.8856	0.9694	0.9664	0.9694	0.7719	0.8591
HG03	0.8078	0.8286	0.8286	0.8700	0.9556	0.9680	0.9556	0.7644	0.8520
NN01	0.2943	0.3036	0.3036	0.2926	0.3043	0.3297	0.3043	0.3293	0.3166
RAW01	0.8053	0.8098	0.8098	0.8384	0.8054	0.7956	0.8054	0.7938	0.8307
RAW02	0.7371	0.7431	0.7431	0.8384	0.8054	0.7774	0.8054	0.7938	0.8307
RD01	0.7558	0.7548	0.7548	0.7373	0.7120	0.6915	0.7120	0.7234	0.7216
RD02	0.7233	0.7297	0.7297	0.8287	0.7636	0.7400	0.7636	0.7628	0.7805
Rumelhart01	0.4919	0.4822	0.4822	0.4917	0.4906	0.5100	0.4906	0.5092	0.5058
Rumelhart02	0.5044	0.4945	0.4945	0.4906	0.4878	0.5147	0.4878	0.5066	0.4983
Server01	0.8002	0.8045	0.8045	0.8686	0.8853	0.8683	0.8853	0.8600	0.9119
Server02	0.8097	0.8235	0.8235	0.8814	0.9103	0.8882	0.9103	0.8299	0.9015
Server03	0.8201	0.8376	0.8376	0.8770	0.9062	0.8884	0.9062	0.8611	0.9225

**ANEXO A.** Tabla de Resultados, utilizando el valor *Sim*. Clasificación Microsoft.

Parámetros  $\theta = 1.00$  y  $\Delta = 0.05$ .

3ª Parte	Ciente03	DMA01	DMA02	DMA03	DMA04	ED01	ED02	ED03	ED04
AMP01	0.8904	0.1183	0.1154	0.1262	0.1209	0.5944	0.6425	0.6783	0.6359
AMP02	0.8183	0.0910	0.0888	0.1026	0.0940	0.4695	0.4998	0.5468	0.5153
AMP03	0.8413	0.0531	0.0518	0.0707	0.0657	0.4462	0.4717	0.5197	0.4927
AMP04	0.8405	0.0484	0.0472	0.0445	0.0598	0.4873	0.5099	0.5537	0.5326
AMP05	0.8339	0.0000	0.0000	0.0000	0.0058	0.4322	0.4570	0.5055	0.4842
AMP06	0.8339	0.0000	0.0000	0.0000	0.0058	0.4322	0.4570	0.5055	0.4842
AMP07	0.8183	0.0910	0.0888	0.1026	0.0940	0.4695	0.4998	0.5468	0.5153
AMP08	0.8840	0.0211	0.0205	0.0342	0.0148	0.5041	0.5375	0.5792	0.5541
AMP09	0.9068	0.0594	0.0579	0.0742	0.0577	0.5694	0.5862	0.6303	0.6124
AMP10	0.8183	0.0910	0.0888	0.1026	0.0940	0.4695	0.4998	0.5468	0.5153
AMP11	0.8286	0.0722	0.0705	0.0896	0.0870	0.4614	0.4919	0.5392	0.5075
AMP12	0.8286	0.0722	0.0705	0.0896	0.0870	0.4614	0.4919	0.5392	0.5075
BW01	0.9100	0.1296	0.1264	0.1377	0.1355	0.6221	0.6714	0.7126	0.6636
BW02	0.9129	0.0711	0.0693	0.0775	0.0640	0.5371	0.5718	0.6163	0.5810
BW03	0.8990	0.0198	0.0193	0.0320	0.0146	0.4881	0.5193	0.5742	0.5421
BW04	0.9129	0.0711	0.0693	0.0775	0.0640	0.5371	0.5718	0.6163	0.5810
Ciente01	0.8730	0.1359	0.1325	0.1592	0.1555	0.6199	0.6257	0.6643	0.6572
Ciente02	0.9396	0.0999	0.0974	0.1290	0.1252	0.6251	0.6262	0.6689	0.6665
Ciente03	1.0000	0.0942	0.0919	0.1186	0.0782	0.5789	0.6003	0.6521	0.6302
DMA01	0.0273	1.0000	0.9773	0.9096	0.8499	0.0372	0.0246	0.0312	0.0361
DMA02	0.0000	0.9767	1.0000	0.8866	0.8250	0.0000	0.0000	0.0000	0.0000
DMA03	0.0415	0.9085	0.8880	1.0000	0.7712	0.0187	0.0132	0.0271	0.0152
DMA04	0.1067	0.8645	0.8460	0.7961	1.0000	0.1373	0.1019	0.1146	0.1426
ED01	0.6911	0.3424	0.3339	0.3381	0.3471	1.0000	0.9575	0.9291	0.9691
ED02	0.6977	0.3131	0.3133	0.3138	0.2992	0.9562	1.0000	0.9702	0.9264
ED03	0.7276	0.2934	0.2888	0.2993	0.2845	0.9243	0.9691	1.0000	0.9574
ED04	0.7210	0.3227	0.3147	0.3166	0.3324	0.9682	0.9266	0.9589	1.0000
GS01	0.9026	0.1104	0.1077	0.1192	0.1103	0.6194	0.6748	0.7158	0.6598
GS02	0.9129	0.1111	0.1083	0.1147	0.1016	0.6306	0.6452	0.6860	0.6706
HG01	0.9133	0.0883	0.0861	0.1009	0.0829	0.5867	0.6267	0.6693	0.6293
HG02	0.8943	0.0253	0.0247	0.0364	0.0134	0.4965	0.5430	0.5886	0.5437
HG03	0.8872	0.0132	0.0129	0.0197	0.0000	0.4727	0.5158	0.5623	0.5207
NN01	0.2884	0.4404	0.4400	0.4359	0.4365	0.1050	0.1307	0.1582	0.1276
RAW01	0.8151	0.2273	0.2288	0.2488	0.2150	0.6876	0.7013	0.7168	0.7017
RAW02	0.8151	0.2273	0.2288	0.2488	0.2150	0.6876	0.7013	0.7168	0.7017
RD01	0.7216	0.2027	0.2109	0.2183	0.1933	0.6164	0.6595	0.6788	0.6395
RD02	0.7688	0.2520	0.2458	0.2480	0.2513	0.7398	0.7811	0.7950	0.7537
Rumelhart01	0.5001	0.4346	0.4284	0.4463	0.4244	0.4897	0.4887	0.5058	0.5034
Rumelhart02	0.5044	0.4528	0.4417	0.4550	0.4516	0.4895	0.4887	0.5058	0.5032
Server01	0.9104	0.1061	0.0977	0.1173	0.1091	0.5837	0.6083	0.6516	0.6263
Server02	0.9186	0.0787	0.0752	0.0934	0.0693	0.5828	0.6168	0.6598	0.6256
Server03	0.9112	0.0738	0.0662	0.0971	0.0684	0.5539	0.5958	0.6395	0.5973

**ANEXO A.** Tabla de Resultados, utilizando el valor *Sim*. Clasificación Microsoft.

Parámetros  $\theta = 1.00$  y  $\Delta = 0.05$ .

4ª Parte	GS01	GS02	HG01	HG02	HG03	NN01	RAW01	RAW02	RD01
AMP01	0.8934	0.8854	0.8936	0.8428	0.8381	0.0750	0.7967	0.7967	0.7588
AMP02	0.8085	0.8275	0.8350	0.8453	0.8226	0.0785	0.7699	0.6894	0.7180
AMP03	0.8104	0.8440	0.8753	0.9016	0.8797	0.0579	0.7502	0.6697	0.6576
AMP04	0.8018	0.8387	0.9095	0.9153	0.8802	0.0369	0.7684	0.7094	0.6366
AMP05	0.7650	0.8070	0.8915	0.9235	0.9007	0.0000	0.7064	0.6474	0.5802
AMP06	0.7650	0.8070	0.8915	0.9235	0.9007	0.0000	0.7064	0.6474	0.5802
AMP07	0.8085	0.8275	0.8350	0.8453	0.8226	0.0785	0.7699	0.6894	0.7180
AMP08	0.8589	0.8903	0.9398	0.9459	0.9459	0.0210	0.7265	0.6969	0.6183
AMP09	0.8980	0.9350	0.9762	0.9386	0.9263	0.0536	0.7663	0.7602	0.6667
AMP10	0.8085	0.8275	0.8350	0.8453	0.8226	0.0785	0.7699	0.6894	0.7180
AMP11	0.8089	0.8279	0.8420	0.8549	0.8386	0.0724	0.7708	0.6903	0.7111
AMP12	0.8089	0.8279	0.8420	0.8549	0.8386	0.0724	0.7708	0.6903	0.7111
BW01	0.9446	0.9349	0.9220	0.8975	0.8850	0.1144	0.8169	0.8169	0.7092
BW02	0.8909	0.9279	0.9552	0.9708	0.9581	0.0721	0.7652	0.7652	0.6603
BW03	0.8670	0.8907	0.9298	0.9662	0.9682	0.0580	0.7400	0.7169	0.6166
BW04	0.8909	0.9279	0.9552	0.9708	0.9581	0.0721	0.7652	0.7652	0.6603
Ciente01	0.8151	0.8148	0.8185	0.7971	0.7930	0.1663	0.7680	0.7680	0.6960
Ciente02	0.8785	0.8781	0.8763	0.8697	0.8647	0.1162	0.8019	0.8019	0.6816
Ciente03	0.9008	0.9113	0.9138	0.9016	0.8962	0.0739	0.7823	0.7823	0.6796
DMA01	0.0278	0.0278	0.0271	0.0254	0.0251	0.2180	0.0229	0.0229	0.0147
DMA02	0.0000	0.0000	0.0000	0.0000	0.0000	0.1975	0.0000	0.0000	0.0000
DMA03	0.0252	0.0195	0.0285	0.0243	0.0192	0.2017	0.0380	0.0380	0.0217
DMA04	0.1225	0.1133	0.1169	0.1097	0.1084	0.2894	0.1041	0.1041	0.1003
ED01	0.7159	0.7240	0.6988	0.6561	0.6442	0.1457	0.7302	0.7302	0.6762
ED02	0.7497	0.7267	0.7195	0.6782	0.6631	0.1445	0.7340	0.7340	0.7037
ED03	0.7735	0.7495	0.7427	0.7000	0.6846	0.1420	0.7388	0.7388	0.7105
ED04	0.7387	0.7468	0.7220	0.6794	0.6672	0.1432	0.7349	0.7349	0.6869
GS01	1.0000	0.9493	0.9360	0.8783	0.8609	0.0343	0.8021	0.8021	0.7095
GS02	0.9493	1.0000	0.9572	0.9138	0.8959	0.0478	0.8074	0.8074	0.7099
HG01	0.9344	0.9561	1.0000	0.9479	0.9361	0.0660	0.7865	0.7865	0.6861
HG02	0.8670	0.9057	0.9444	1.0000	0.9876	0.0494	0.7205	0.7205	0.6104
HG03	0.8461	0.8848	0.9310	0.9874	1.0000	0.0268	0.7000	0.7000	0.6022
NN01	0.2448	0.2548	0.2869	0.3199	0.3120	1.0000	0.3232	0.3180	0.1631
RAW01	0.8290	0.8334	0.8199	0.7790	0.7656	0.2521	1.0000	1.0000	0.7112
RAW02	0.8290	0.8334	0.8199	0.7790	0.7656	0.2463	1.0000	1.0000	0.7112
RD01	0.7431	0.7433	0.7290	0.6848	0.6820	0.0536	0.7045	0.7045	1.0000
RD02	0.8411	0.8205	0.8036	0.7450	0.7408	0.1564	0.7994	0.7994	0.7455
Rumelhart01	0.4988	0.4923	0.4948	0.4884	0.4875	0.4193	0.5373	0.5373	0.4810
Rumelhart02	0.4995	0.4925	0.4917	0.4908	0.4898	0.4189	0.5346	0.5346	0.4783
Server01	0.8983	0.8763	0.8931	0.8674	0.8509	0.1354	0.7800	0.7800	0.6681
Server02	0.8991	0.8772	0.9055	0.8832	0.8760	0.0370	0.7752	0.7752	0.6786
Server03	0.8925	0.8706	0.9079	0.8813	0.8762	0.0522	0.7754	0.7754	0.6744

**ANEXO A.** Tabla de Resultados, utilizando el valor *Sim*. Clasificación Microsoft.

Parámetros  $\theta = 1.00$  y  $\Delta = 0.05$ .

5ª Parte	RD02	Rumelhart01	Rumelhart02	Server01	Server02	Server03
AMP01	0.7842	0.1521	0.1514	0.8670	0.8657	0.8810
AMP02	0.6657	0.1899	0.1911	0.7982	0.8125	0.8244
AMP03	0.6578	0.1415	0.1255	0.8033	0.8270	0.8267
AMP04	0.6863	0.1187	0.0977	0.7636	0.7870	0.7892
AMP05	0.6277	0.0791	0.0395	0.7080	0.7422	0.7448
AMP06	0.6277	0.0791	0.0395	0.7080	0.7422	0.7448
AMP07	0.6657	0.1899	0.1911	0.7982	0.8125	0.8244
AMP08	0.6620	0.1492	0.1115	0.8317	0.8597	0.8551
AMP09	0.7295	0.1956	0.1773	0.8653	0.8787	0.8822
AMP10	0.6657	0.1899	0.1911	0.7982	0.8125	0.8244
AMP11	0.6668	0.1578	0.1583	0.7986	0.8226	0.8384
AMP12	0.6668	0.1578	0.1583	0.7986	0.8226	0.8384
BW01	0.8015	0.2231	0.2029	0.8728	0.8880	0.8849
BW02	0.7082	0.1705	0.1461	0.8816	0.9097	0.9065
BW03	0.6620	0.1592	0.1474	0.8569	0.8814	0.8827
BW04	0.7082	0.1705	0.1461	0.8816	0.9097	0.9065
Ciente01	0.7272	0.2551	0.2333	0.8654	0.8404	0.8710
Ciente02	0.7374	0.2196	0.1889	0.9119	0.9039	0.9251
Ciente03	0.7217	0.2057	0.1937	0.9098	0.9201	0.9136
DMA01	0.0328	0.0352	0.0440	0.0339	0.0285	0.0327
DMA02	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
DMA03	0.0153	0.0433	0.0359	0.0339	0.0319	0.0451
DMA04	0.1264	0.1137	0.1355	0.1310	0.1143	0.1220
ED01	0.7702	0.4052	0.3909	0.6927	0.6995	0.6818
ED02	0.8006	0.3857	0.3710	0.7018	0.7155	0.7027
ED03	0.8066	0.3850	0.3703	0.7253	0.7384	0.7254
ED04	0.7762	0.4046	0.3902	0.7162	0.7226	0.7045
GS01	0.8120	0.2175	0.1999	0.8994	0.9027	0.8973
GS02	0.7877	0.2079	0.1894	0.8778	0.8816	0.8764
HG01	0.7620	0.1922	0.1678	0.8917	0.9066	0.9099
HG02	0.6703	0.1270	0.1103	0.8566	0.8768	0.8760
HG03	0.6608	0.1148	0.0979	0.8369	0.8676	0.8691
NN01	0.2194	0.2911	0.2737	0.3313	0.2734	0.2917
RAW01	0.7949	0.3757	0.3570	0.8120	0.8125	0.8145
RAW02	0.7949	0.3757	0.3570	0.8120	0.8125	0.8145
RD01	0.7337	0.2835	0.2626	0.7098	0.7258	0.7248
RD02	1.0000	0.3732	0.3330	0.7753	0.7725	0.7650
Rumelhart01	0.5250	1.0000	0.9159	0.5173	0.5075	0.5155
Rumelhart02	0.5063	0.9179	1.0000	0.5069	0.4997	0.5102
Server01	0.7312	0.2379	0.2030	1.0000	0.9303	0.9431
Server02	0.7210	0.2030	0.1713	0.9285	1.0000	0.9455
Server03	0.7090	0.2084	0.1808	0.9411	0.9450	1.0000

**ANEXO B.** Tabla de Resultados, utilizando la distancia FDTW. Clasificación Microsoft. Parámetros  $\theta = 1.00$  y  $\Delta = 0.05$ .

1ª Parte	AMP01	AMP02	AMP03	AMP04	AMP05	AMP06	AMP07	AMP08	AMP09
AMP01	0.00	38.00	49.55	54.40	69.20	69.20	38.00	70.00	55.20
AMP02	38.00	0.00	27.40	41.55	63.55	63.55	0.00	82.55	60.55
AMP03	49.55	27.40	0.00	17.15	39.15	39.15	27.40	58.15	36.15
AMP04	54.40	41.55	17.15	0.00	14.80	14.80	41.55	26.60	11.80
AMP05	69.20	63.55	39.15	14.80	0.00	0.00	63.55	11.80	19.90
AMP06	69.20	63.55	39.15	14.80	0.00	0.00	63.55	11.80	19.90
AMP07	38.00	0.00	27.40	41.55	63.55	63.55	0.00	82.55	60.55
AMP08	70.00	82.55	58.15	26.60	11.80	11.80	82.55	0.00	14.80
AMP09	55.20	60.55	36.15	11.80	19.90	19.90	60.55	14.80	0.00
AMP10	38.00	0.00	27.40	41.55	63.55	63.55	0.00	82.55	60.55
AMP11	28.50	9.50	21.05	36.70	58.70	58.70	9.50	77.70	55.70
AMP12	28.50	9.50	21.05	36.70	58.70	58.70	9.50	77.70	55.70
BW01	32.40	78.70	71.70	57.40	71.50	71.50	78.70	54.25	39.00
BW02	66.95	65.45	45.15	46.00	50.50	50.50	65.45	39.10	39.85
BW03	76.20	69.70	46.10	41.45	30.25	30.25	69.70	24.00	29.65
BW04	67.85	65.45	46.65	46.00	50.50	50.50	65.45	40.60	39.85
Ciente01	64.35	109.70	122.15	123.95	132.60	132.60	109.70	115.75	96.55
Ciente02	58.90	89.75	89.85	89.55	99.70	99.70	89.75	79.50	62.15
Ciente03	51.65	85.60	74.75	75.15	78.25	78.25	85.60	54.65	43.90
DMA01	446.05	459.85	479.05	481.40	505.90	505.90	459.85	495.25	475.85
DMA02	458.90	472.70	491.90	494.25	518.75	518.75	472.70	508.10	488.70
DMA03	447.65	459.75	476.10	489.50	512.30	512.30	459.75	494.80	474.30
DMA04	401.35	413.65	426.55	429.25	453.90	453.90	413.65	449.80	430.20
ED01	140.15	183.30	191.35	177.15	196.20	196.20	183.30	171.35	148.80
ED02	127.35	178.20	188.20	174.60	193.45	193.45	178.20	164.75	147.40
ED03	118.70	167.20	177.20	164.65	182.45	182.45	167.20	155.25	136.40
ED04	129.45	172.30	180.35	166.15	183.35	183.35	172.30	158.50	137.80
GS01	49.35	88.65	87.75	91.75	108.80	108.80	88.65	65.30	47.20
GS02	53.00	79.80	72.15	74.60	89.25	89.25	79.80	50.75	30.05
HG01	50.45	78.25	59.10	42.90	51.45	51.45	78.25	28.55	11.30
HG02	79.55	78.25	49.80	42.85	38.70	38.70	78.25	27.35	31.05
HG03	82.90	90.85	61.60	61.35	50.85	50.85	90.85	27.70	37.75
NN01	334.85	333.60	341.05	348.65	362.00	362.00	333.60	354.40	342.60
RAW01	81.35	92.05	99.95	92.65	117.45	117.45	92.05	109.40	93.50
RAW02	81.35	124.25	132.15	116.25	141.05	141.05	124.25	121.25	95.95
RD01	98.75	115.45	140.15	148.75	171.85	171.85	115.45	156.25	136.45
RD02	84.45	130.80	133.90	122.75	145.65	145.65	130.80	132.25	105.85
Rumelhart01	251.40	240.20	254.55	261.30	273.05	273.05	240.20	252.25	238.50
Rumelhart02	245.75	234.25	253.25	261.30	278.15	278.15	234.25	257.30	238.25
Server01	62.25	94.45	92.05	110.65	136.65	136.65	94.45	78.75	63.05
Server02	64.45	89.95	83.00	102.20	123.70	123.70	89.95	67.30	58.20
Server03	57.65	85.05	83.95	102.10	123.60	123.60	85.05	70.20	57.05

**ANEXO B.** Tabla de Resultados, utilizando la distancia FDTW. Clasificación Microsoft. Parámetros  $\theta = 1.00$  y  $\Delta = 0.05$ .

2ª Parte	AMP10	AMP11	AMP12	BW01	BW02	BW03	BW04	Ciente01	Ciente02
AMP01	38.00	28.50	28.50	32.40	66.95	76.20	67.85	64.35	58.90
AMP02	0.00	9.50	9.50	78.70	65.45	69.70	65.45	109.70	89.75
AMP03	27.40	21.05	21.05	71.70	45.15	46.10	46.65	122.15	89.85
AMP04	41.55	36.70	36.70	57.40	46.00	41.45	46.00	123.95	89.55
AMP05	63.55	58.70	58.70	71.50	50.50	30.25	50.50	132.60	99.70
AMP06	63.55	58.70	58.70	71.50	50.50	30.25	50.50	132.60	99.70
AMP07	0.00	9.50	9.50	78.70	65.45	69.70	65.45	109.70	89.75
AMP08	82.55	77.70	77.70	54.25	39.10	24.00	40.60	115.75	79.50
AMP09	60.55	55.70	55.70	39.00	39.85	29.65	39.85	96.55	62.15
AMP10	0.00	9.50	9.50	78.70	65.45	69.70	65.45	109.70	89.75
AMP11	9.50	0.00	0.00	77.20	58.55	64.85	58.55	108.85	89.95
AMP12	9.50	0.00	0.00	77.20	58.55	64.85	58.55	108.85	89.95
BW01	78.70	77.20	77.20	0.00	39.05	56.35	39.05	72.90	45.50
BW02	65.45	58.55	58.55	39.05	0.00	21.85	0.00	88.45	52.70
BW03	69.70	64.85	64.85	56.35	21.85	0.00	22.90	100.70	66.30
BW04	65.45	58.55	58.55	39.05	0.00	22.90	0.00	88.45	52.70
Ciente01	109.70	108.85	108.85	72.90	88.45	100.70	88.45	0.00	38.10
Ciente02	89.75	89.95	89.95	45.50	52.70	66.30	52.70	38.10	0.00
Ciente03	85.60	80.75	80.75	42.40	41.05	47.60	41.05	59.85	28.45
DMA01	459.85	469.35	469.35	440.35	469.95	495.90	469.95	437.15	455.35
DMA02	472.70	482.20	482.20	453.20	482.80	508.75	482.80	450.00	468.20
DMA03	459.75	466.40	466.40	441.75	472.60	495.90	472.60	430.75	446.20
DMA04	413.65	416.85	416.85	394.70	427.35	449.90	427.35	385.55	399.40
ED01	183.30	186.10	186.10	130.60	159.95	176.90	159.95	131.35	129.55
ED02	178.20	181.00	181.00	117.05	152.55	171.25	152.55	133.35	133.15
ED03	167.20	170.00	170.00	106.05	141.55	157.10	141.55	123.85	122.15
ED04	172.30	175.10	175.10	119.60	148.95	162.80	148.95	121.85	118.55
GS01	88.65	88.45	88.45	25.65	50.50	61.55	50.50	85.60	56.25
GS02	79.80	79.60	79.60	30.10	33.35	50.55	33.35	85.65	56.40
HG01	78.25	74.90	74.90	37.00	21.25	33.30	21.25	86.05	58.65
HG02	78.25	73.40	73.40	51.85	14.75	17.10	14.75	102.65	65.95
HG03	90.85	82.65	82.65	58.90	21.45	16.30	21.45	106.00	69.30
NN01	333.60	335.80	335.80	320.60	335.90	341.00	335.90	301.80	319.95
RAW01	92.05	91.70	91.70	73.25	93.95	104.00	93.95	92.80	79.25
RAW02	124.25	123.90	123.90	73.25	93.95	113.25	93.95	92.80	79.25
RD01	115.45	118.25	118.25	119.05	139.05	156.95	139.05	124.45	130.35
RD02	130.80	130.35	130.35	77.65	114.15	132.25	114.15	106.75	102.75
Rumelhart01	240.20	249.70	249.70	230.35	245.95	249.30	245.95	220.85	231.40
Rumelhart02	234.25	243.75	243.75	230.85	247.30	246.90	247.30	222.05	234.90
Server01	94.45	94.25	94.25	59.55	55.40	67.00	55.40	63.00	41.25
Server02	89.95	85.10	85.10	53.75	43.30	56.90	43.30	76.55	46.10
Server03	85.05	78.30	78.30	55.75	45.30	56.80	45.30	62.50	36.30

**ANEXO B.** Tabla de Resultados, utilizando la distancia FDTW. Clasificación Microsoft. Parámetros  $\theta = 1.00$  y  $\Delta = 0.05$ .

3ª Parte	Ciente03	DMA01	DMA02	DMA03	DMA04	ED01	ED02	ED03	ED04
AMP01	51.65	446.05	458.90	447.65	401.35	140.15	127.35	118.70	129.45
AMP02	85.60	459.85	472.70	459.75	413.65	183.30	178.20	167.20	172.30
AMP03	74.75	479.05	491.90	476.10	426.55	191.35	188.20	177.20	180.35
AMP04	75.15	481.40	494.25	489.50	429.25	177.15	174.60	164.65	166.15
AMP05	78.25	505.90	518.75	512.30	453.90	196.20	193.45	182.45	183.35
AMP06	78.25	505.90	518.75	512.30	453.90	196.20	193.45	182.45	183.35
AMP07	85.60	459.85	472.70	459.75	413.65	183.30	178.20	167.20	172.30
AMP08	54.65	495.25	508.10	494.80	449.80	171.35	164.75	155.25	158.50
AMP09	43.90	475.85	488.70	474.30	430.20	148.80	147.40	136.40	137.80
AMP10	85.60	459.85	472.70	459.75	413.65	183.30	178.20	167.20	172.30
AMP11	80.75	469.35	482.20	466.40	416.85	186.10	181.00	170.00	175.10
AMP12	80.75	469.35	482.20	466.40	416.85	186.10	181.00	170.00	175.10
BW01	42.40	440.35	453.20	441.75	394.70	130.60	117.05	106.05	119.60
BW02	41.05	469.95	482.80	472.60	427.35	159.95	152.55	141.55	148.95
BW03	47.60	495.90	508.75	495.90	449.90	176.90	171.25	157.10	162.80
BW04	41.05	469.95	482.80	472.60	427.35	159.95	152.55	141.55	148.95
Ciente01	59.85	437.15	450.00	430.75	385.55	131.35	133.35	123.85	121.85
Ciente02	28.45	455.35	468.20	446.20	399.40	129.55	133.15	122.15	118.55
Ciente03	0.00	458.25	471.10	451.55	420.85	145.50	142.40	128.35	131.45
DMA01	458.25	0.00	11.80	46.30	68.55	332.70	347.50	357.45	342.65
DMA02	471.10	11.80	0.00	58.10	79.90	345.55	356.25	368.95	355.50
DMA03	451.55	46.30	58.10	0.00	104.45	339.10	351.55	358.95	350.10
DMA04	420.85	68.55	79.90	104.45	0.00	298.10	319.95	326.65	304.80
ED01	145.50	332.70	345.55	339.10	298.10	0.00	15.15	26.15	11.00
ED02	142.40	347.50	356.25	351.55	319.95	15.15	0.00	11.00	26.15
ED03	128.35	357.45	368.95	358.95	326.65	26.15	11.00	0.00	15.15
ED04	131.45	342.65	355.50	350.10	304.80	11.00	26.15	15.15	0.00
GS01	45.90	450.05	462.90	451.25	406.20	131.50	115.85	104.85	120.95
GS02	41.05	449.70	462.55	453.55	410.15	127.65	126.40	115.85	117.10
HG01	40.85	461.25	474.10	460.60	418.70	142.80	133.00	122.00	131.80
HG02	49.80	493.10	505.95	493.65	450.45	174.00	162.80	151.80	162.20
HG03	53.15	499.20	512.05	502.20	456.55	182.20	172.50	161.50	170.40
NN01	335.25	283.10	290.50	289.00	257.25	309.25	309.70	310.60	310.15
RAW01	87.10	390.90	400.05	384.85	358.40	107.95	106.40	104.50	106.05
RAW02	87.10	390.90	400.05	384.85	358.40	107.95	106.40	104.50	106.05
RD01	131.15	403.35	409.35	400.45	368.30	132.55	121.30	118.50	128.15
RD02	108.90	378.40	391.25	385.25	341.80	89.90	78.00	75.65	87.55
Rumelhart01	235.50	286.05	296.50	283.65	262.80	176.35	182.15	182.35	176.55
Rumelhart02	233.50	276.85	289.60	279.20	250.35	176.40	182.15	182.35	176.60
Server01	42.20	452.20	468.05	452.20	406.75	143.85	139.55	128.55	132.85
Server02	38.35	466.10	479.75	464.45	424.90	144.15	136.50	125.50	133.10
Server03	41.85	468.55	484.40	462.55	425.30	154.15	144.00	133.00	143.15

**ANEXO B.** Tabla de Resultados, utilizando la distancia FDTW. Clasificación Microsoft. Parámetros  $\theta = 1.00$  y  $\Delta = 0.05$ .

4ª Parte	GS01	GS02	HG01	HG02	HG03	NN01	RAW01	RAW02	RD01
AMP01	49.35	53.00	50.45	79.55	82.90	334.85	81.35	81.35	98.75
AMP02	88.65	79.80	78.25	78.25	90.85	333.60	92.05	124.25	115.45
AMP03	87.75	72.15	59.10	49.80	61.60	341.05	99.95	132.15	140.15
AMP04	91.75	74.60	42.90	42.85	61.35	348.65	92.65	116.25	148.75
AMP05	108.80	89.25	51.45	38.70	50.85	362.00	117.45	141.05	171.85
AMP06	108.80	89.25	51.45	38.70	50.85	362.00	117.45	141.05	171.85
AMP07	88.65	79.80	78.25	78.25	90.85	333.60	92.05	124.25	115.45
AMP08	65.30	50.75	28.55	27.35	27.70	354.40	109.40	121.25	156.25
AMP09	47.20	30.05	11.30	31.05	37.75	342.60	93.50	95.95	136.45
AMP10	88.65	79.80	78.25	78.25	90.85	333.60	92.05	124.25	115.45
AMP11	88.45	79.60	74.90	73.40	82.65	335.80	91.70	123.90	118.25
AMP12	88.45	79.60	74.90	73.40	82.65	335.80	91.70	123.90	118.25
BW01	25.65	30.10	37.00	51.85	58.90	320.60	73.25	73.25	119.05
BW02	50.50	33.35	21.25	14.75	21.45	335.90	93.95	93.95	139.05
BW03	61.55	50.55	33.30	17.10	16.30	341.00	104.00	113.25	156.95
BW04	50.50	33.35	21.25	14.75	21.45	335.90	93.95	93.95	139.05
Ciente01	85.60	85.65	86.05	102.65	106.00	301.80	92.80	92.80	124.45
Ciente02	56.25	56.40	58.65	65.95	69.30	319.95	79.25	79.25	130.35
Ciente03	45.90	41.05	40.85	49.80	53.15	335.25	87.10	87.10	131.15
DMA01	450.05	449.70	461.25	493.10	499.20	283.10	390.90	390.90	403.35
DMA02	462.90	462.55	474.10	505.95	512.05	290.50	400.05	400.05	409.35
DMA03	451.25	453.55	460.60	493.65	502.20	289.00	384.85	384.85	400.45
DMA04	406.20	410.15	418.70	450.45	456.55	257.25	358.40	358.40	368.30
ED01	131.50	127.65	142.80	174.00	182.20	309.25	107.95	107.95	132.55
ED02	115.85	126.40	133.00	162.80	172.50	309.70	106.40	106.40	121.30
ED03	104.85	115.85	122.00	151.80	161.50	310.60	104.50	104.50	118.50
ED04	120.95	117.10	131.80	162.20	170.40	310.15	106.05	106.05	128.15
GS01	0.00	23.45	30.35	61.55	71.25	349.60	79.15	79.15	118.90
GS02	23.45	0.00	20.30	43.60	53.30	344.70	77.05	77.05	118.75
HG01	30.35	20.30	0.00	26.35	32.70	338.10	85.40	85.40	128.50
HG02	61.55	43.60	26.35	0.00	6.35	344.10	111.80	111.80	159.50
HG03	71.25	53.30	32.70	6.35	0.00	352.30	120.00	120.00	162.85
NN01	349.60	344.70	338.10	344.10	352.30	0.00	270.75	272.85	342.60
RAW01	79.15	77.05	85.40	111.80	120.00	270.75	0.00	0.00	118.20
RAW02	79.15	77.05	85.40	111.80	120.00	272.85	0.00	0.00	118.20
RD01	118.90	118.75	128.50	159.50	162.85	342.60	118.20	118.20	0.00
RD02	73.55	83.05	93.10	129.00	132.70	305.40	80.25	80.25	104.20
Rumelhart01	232.00	234.85	239.50	258.85	262.45	210.20	185.10	185.10	212.45
Rumelhart02	231.70	234.75	241.00	257.65	261.25	210.35	186.20	186.20	213.55
Server01	47.10	57.20	50.70	67.10	76.35	313.00	88.00	88.00	135.85
Server02	46.70	56.80	44.80	59.10	63.50	348.60	89.95	89.95	131.55
Server03	49.75	59.85	43.65	60.05	63.40	343.10	89.85	89.85	133.30



**ANEXO B.** Tabla de Resultados, utilizando la distancia FDTW. Clasificación Microsoft. Parámetros  $\theta = 1.00$  y  $\Delta = 0.05$ .

5ª Parte	RD02	Rumelhart01	Rumelhart02	Server01	Server02	Server03
AMP01	84.45	251.40	245.75	62.25	64.45	57.65
AMP02	130.80	240.20	234.25	94.45	89.95	85.05
AMP03	133.90	254.55	253.25	92.05	83.00	83.95
AMP04	122.75	261.30	261.30	110.65	102.20	102.10
AMP05	145.65	273.05	278.15	136.65	123.70	123.60
AMP06	145.65	273.05	278.15	136.65	123.70	123.60
AMP07	130.80	240.20	234.25	94.45	89.95	85.05
AMP08	132.25	252.25	257.30	78.75	67.30	70.20
AMP09	105.85	238.50	238.25	63.05	58.20	57.05
AMP10	130.80	240.20	234.25	94.45	89.95	85.05
AMP11	130.35	249.70	243.75	94.25	85.10	78.30
AMP12	130.35	249.70	243.75	94.25	85.10	78.30
BW01	77.65	230.35	230.85	59.55	53.75	55.75
BW02	114.15	245.95	247.30	55.40	43.30	45.30
BW03	132.25	249.30	246.90	67.00	56.90	56.80
BW04	114.15	245.95	247.30	55.40	43.30	45.30
Ciente01	106.75	220.85	222.05	63.00	76.55	62.50
Ciente02	102.75	231.40	234.90	41.25	46.10	36.30
Ciente03	108.90	235.50	233.50	42.20	38.35	41.85
DMA01	378.40	286.05	276.85	452.20	466.10	468.55
DMA02	391.25	296.50	289.60	468.05	479.75	484.40
DMA03	385.25	283.65	279.20	452.20	464.45	462.55
DMA04	341.80	262.80	250.35	406.75	424.90	425.30
ED01	89.90	176.35	176.40	143.85	144.15	154.15
ED02	78.00	182.15	182.15	139.55	136.50	144.00
ED03	75.65	182.35	182.35	128.55	125.50	133.00
ED04	87.55	176.55	176.60	132.85	133.10	143.15
GS01	73.55	232.00	231.70	47.10	46.70	49.75
GS02	83.05	234.85	234.75	57.20	56.80	59.85
HG01	93.10	239.50	241.00	50.70	44.80	43.65
HG02	129.00	258.85	257.65	67.10	59.10	60.05
HG03	132.70	262.45	261.25	76.35	63.50	63.40
NN01	305.40	210.20	210.35	313.00	348.60	343.10
RAW01	80.25	185.10	186.20	88.00	89.95	89.85
RAW02	80.25	185.10	186.20	88.00	89.95	89.85
RD01	104.20	212.45	213.55	135.85	131.55	133.30
RD02	0.00	185.85	193.15	105.15	109.15	113.85
Rumelhart01	185.85	0.00	24.35	225.95	236.30	234.70
Rumelhart02	193.15	24.35	0.00	230.80	240.00	237.25
Server01	105.15	225.95	230.80	0.00	33.45	27.55
Server02	109.15	236.30	240.00	33.45	0.00	26.40
Server03	113.85	234.70	237.25	27.55	26.40	0.00

## Otros Experimentos

En esta sección, se muestran diversos resultados aplicando el algoritmo de identificación de subsecuencias propuesto en el Capítulo 3 de esta tesis, a Series de Tiempo (ST) de Origen Natural y Artificial con modificaciones sistemáticas, estas ST se encuentran descritas al final de los experimentos. Algunos de estos resultados se encuentran en [Mirón-Bernal06].

Dadas dos Secuencias  $Q$  y  $C$ , (Ec. 3.2), al aplicar el método propuesto para identificar subsecuencias, se obtienen dos secuencias propias a cada Serie de Tiempo por cada subsecuencia identificada, esto es, la subsecuencia  $sI$  definida contiene parte de la secuencia original de  $Q$ , y parte de la secuencia similar correspondiente a  $C$ .

Para mostrar los resultados a continuación, se obtuvo el promedio aritmético entre estas secuencias propias de la subsecuencia, para mostrar una subsecuencia *similar* a ambas Secuencias  $Q$  y  $C$  de entrada.

Dada una subsecuencia  $sI$ , de longitud  $z$  se obtiene:

$$sI [sub(Q)] = \{ q_1, q_2, q_i \dots q_z \}, \forall q \in Q$$

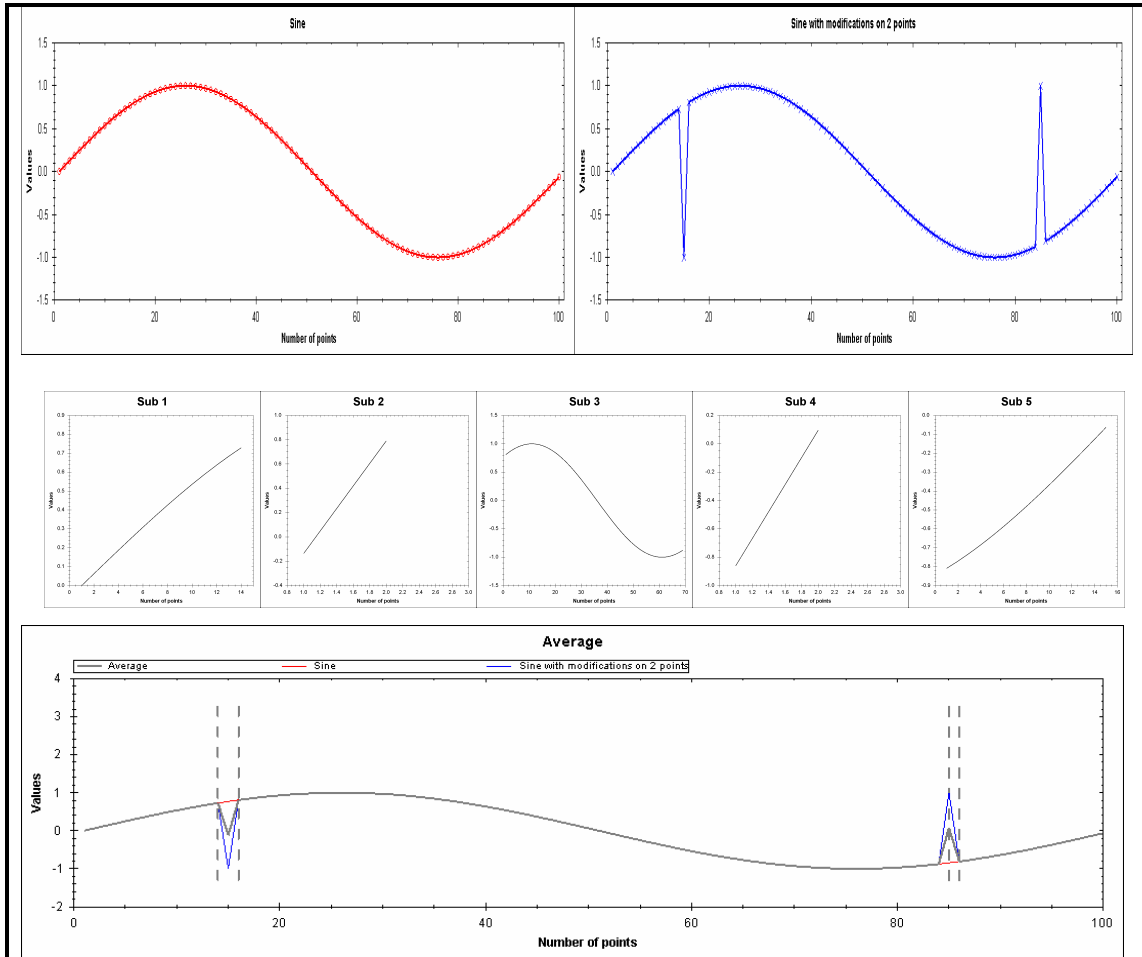
$$sI [sub(C)] = \{ c_1, c_2, c_j \dots c_z \}, \forall c \in C$$

$$sI(Q, C) = \{ prom(q_1, c_1), prom(q_i, c_j), \dots, prom(q_z, c_z) \} \quad 1 \leq i, j \leq z$$

$$prom(q_i, c_j) = (q_i + c_j) / 2$$

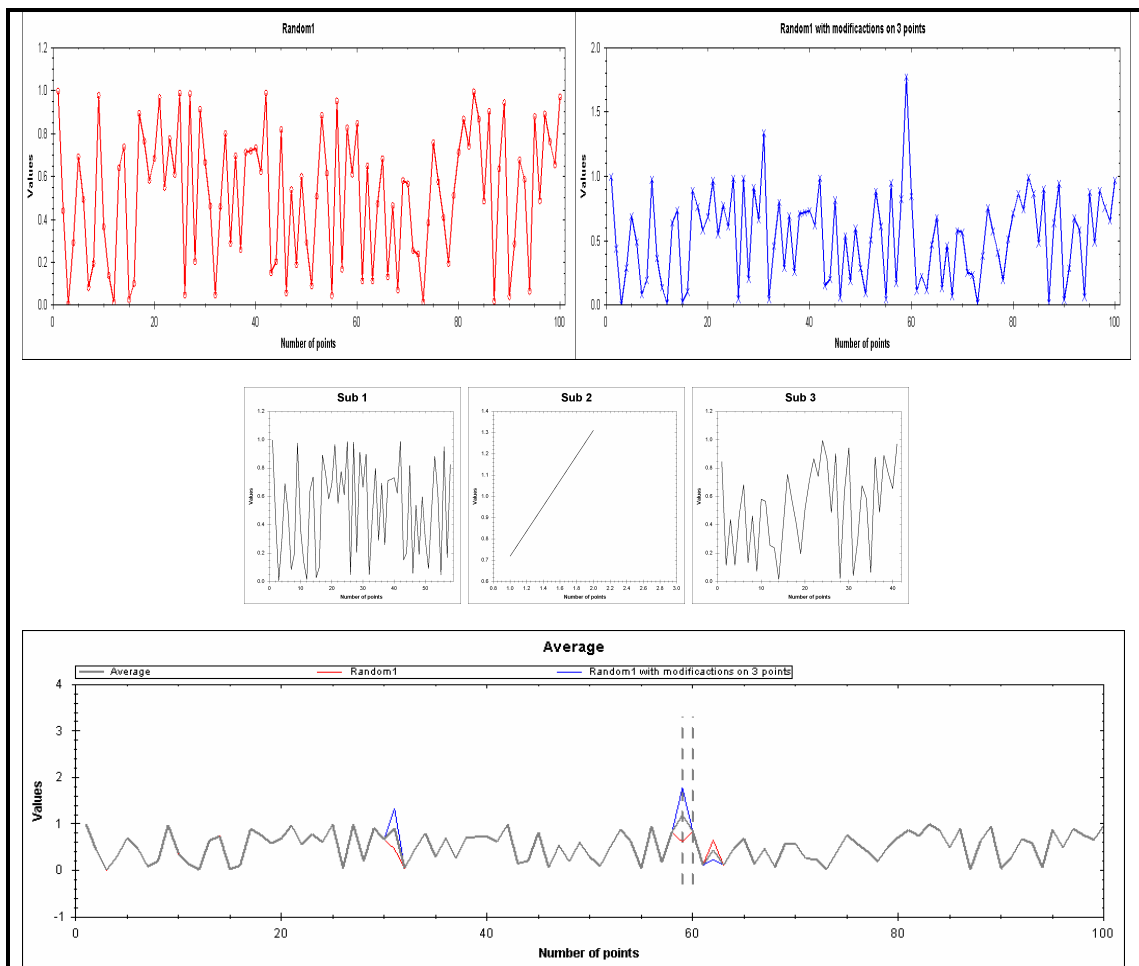
En la Figura A.1, se muestra una gráfica de la función Seno (rojo), limitada a 100 puntos. En color azul, se muestra la función Seno, modificada en dos puntos escogidos de forma aleatoria por el generador de números aleatorios *MersenneTwister*. [Matsumoto98]

Al aplicar el algoritmo propuesto para la identificación de subsecuencias, obtuvimos cinco subsecuencias similares. En la Figura A.1 se muestran los resultados. Para este experimento  $FDTW = 3.6084$ .



**Figura A.1.** Serie de Tiempo *Sine* limitada a cien puntos (rojo), Serie de Tiempo *Sine* modificada en dos puntos (azul), subsecuencias comunes a ambas Series de Tiempo.

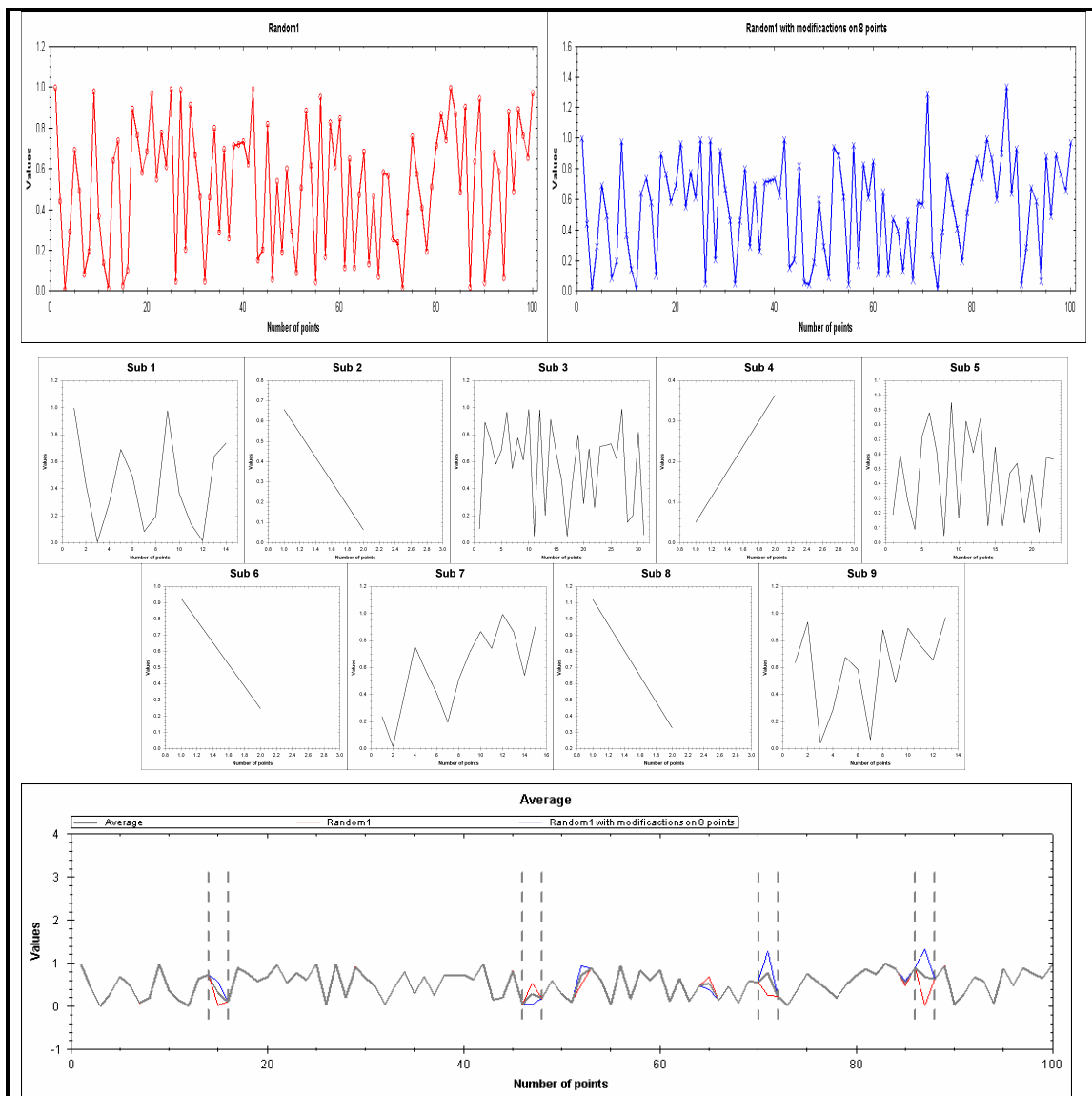
En la A.2. se muestra la Serie de Tiempo (ST) *Random1* (rojo), esta Serie de Tiempo fue modificada en 3 puntos de forma aleatoria utilizando el generador de números aleatorios anteriormente descrito, para obtener la Serie azul.



**Figura A.2.** Serie de Tiempo *Random1* limitada a cien puntos (rojo), Serie de Tiempo *Random1* modificada en 3 puntos (azul), subsecuencias similares detectadas: tres.

De la misma forma que en el experimento anterior, se realizó el promedio aritmético a las subsecuencias obtenidas para obtener secuencias similares entre las Series de Tiempo de entrada. En este caso, el algoritmo muestra tres subsecuencias similares, realizando el “corte” en uno de los puntos que fueron alterados. En la Figura A.2. también se muestra una gráfica del promedio de ambas secuencias donde se observa que los dos puntos restantes alterados en la Serie de Tiempo azul, no fueron valores significativos para mostrar una diferencia entre ambas Series de Tiempo, esto es, los valores modificados eran “propios” de las Series de Tiempo originales. La detección

de subsecuencias similares o patrones en Series de Tiempo, no es una tarea trivial. Al revisar la matriz de distancias en busca de una explicación, se observó que un camino diferente en la construcción del *warp path* de este experimento sería más costoso que el camino actual. El alineamiento entre estas dos Series de Tiempo de entrada mostrada en la Figura A.2, es la que posee el costo mínimo de alineamiento dado por la distancia *Fast Dynamic Time Warping* descrita en el Capítulo 3. Para la Figura A.2  $FDTW = 2.4400$ .



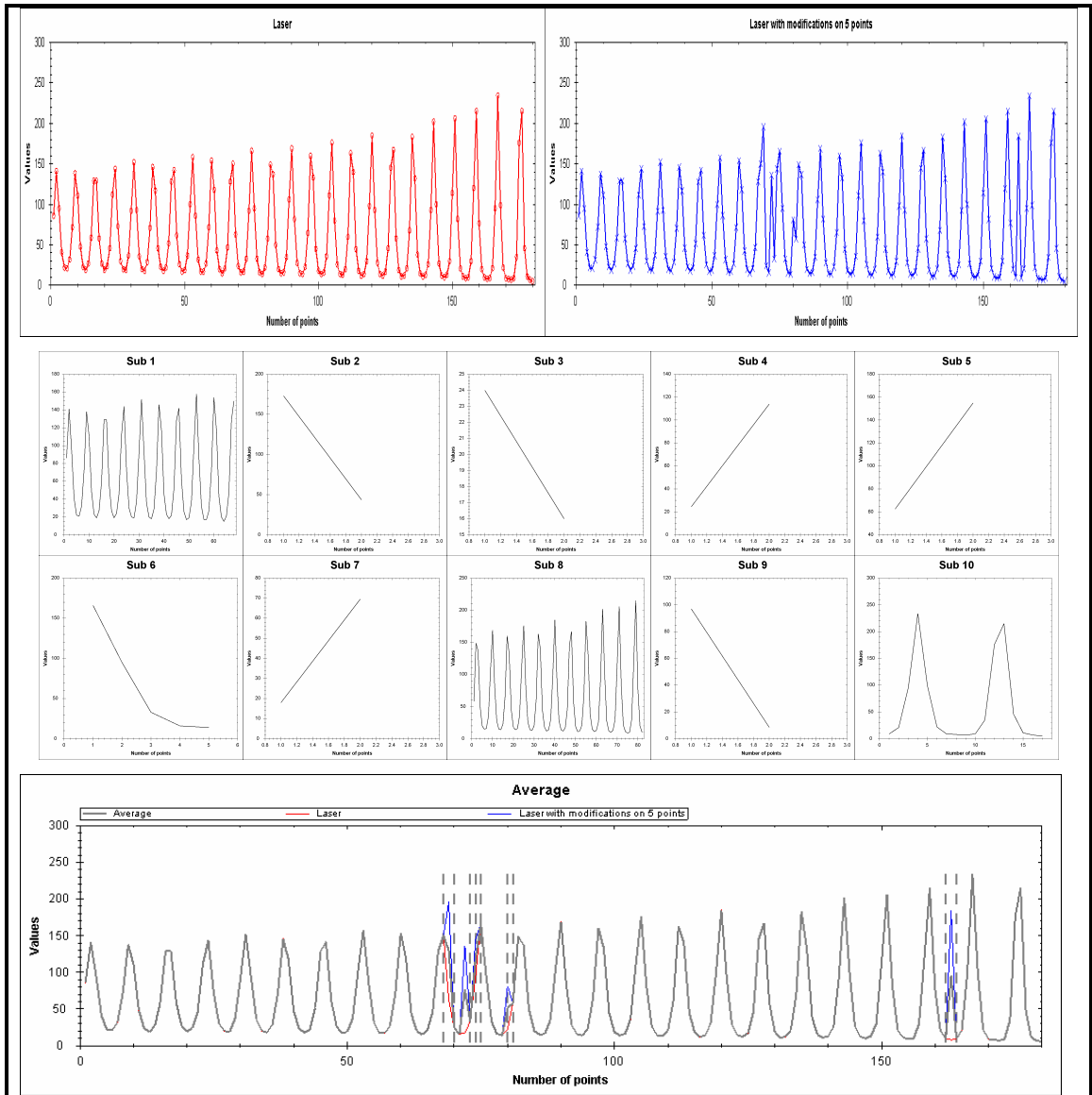
**Figura A.3.** Serie de Tiempo *Random1* (rojo), Serie de Tiempo *Random1 modificada en 8 puntos* (azul), subsecuencias similares detectadas: nueve

En la Figura A.3 se muestra la Serie de Tiempo *Random1* y la Serie que representa ocho modificaciones sistemáticas realizadas sobre la Serie *Random1*. En este experimento se muestra, que los “cortes” realizados por el algoritmo para la identificación de subsecuencias se efectúan en cuatro de los puntos alterados sistemáticamente, obteniendo nueve subsecuencias similares. En la gráfica donde se muestra el promedio de ambas Series de Tiempo, se observa que los cortes fueron realizados donde los valores modificados no eran semejantes a los valores vecinos de la Serie de Tiempo de entrada *Random1*. Para la Figura A.3  $FDTW = 3.2333$ .

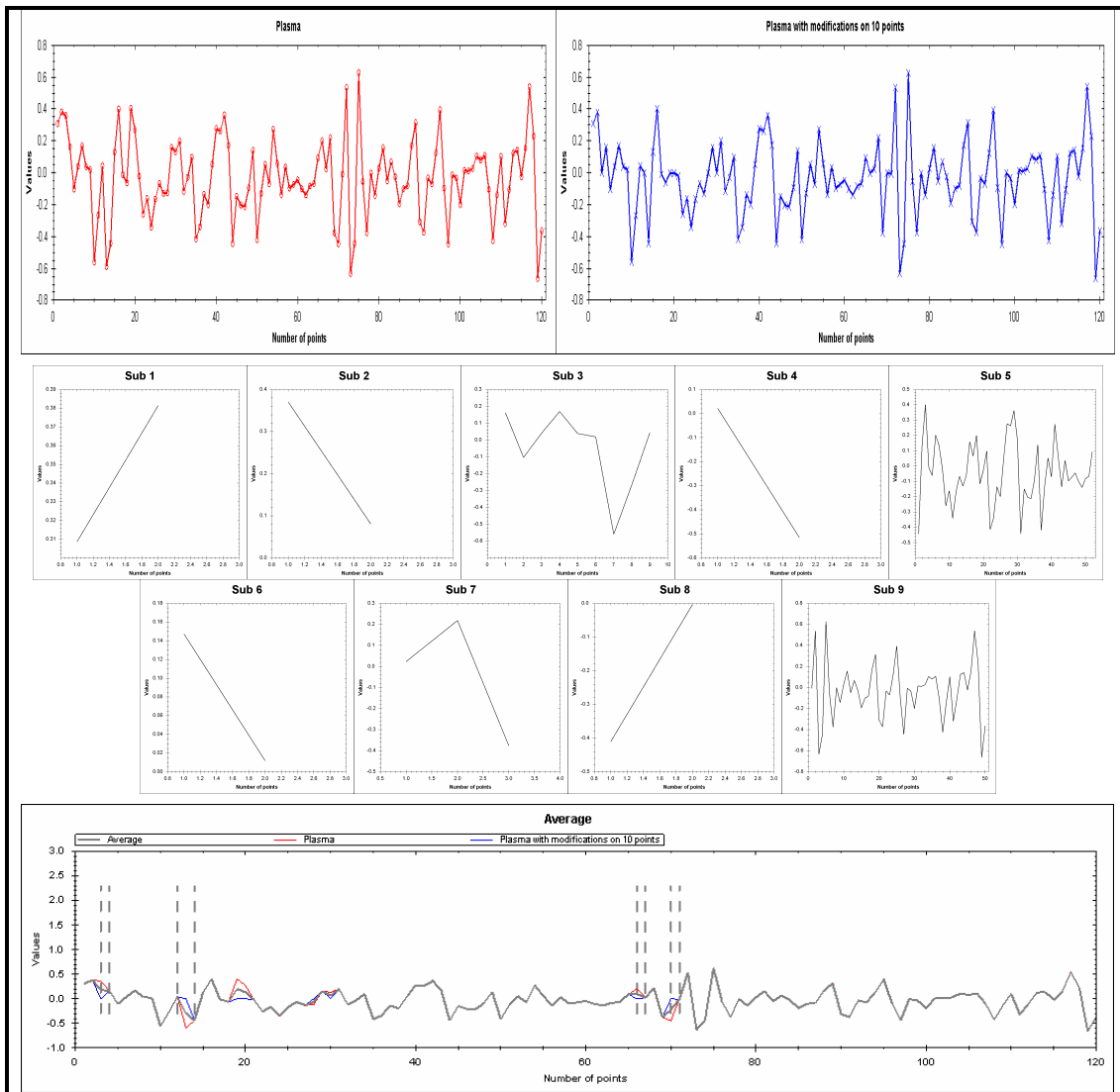
En la Figura A.4. se muestra la ST Láser (rojo) acotada a 180 puntos, y la ST Láser5 que tiene modificaciones sistemáticas en 5 puntos de la ST Láser. Los valores modificados se hicieron en base a una búsqueda de valores máximos y mínimos dentro de la ST Láser. En esta figura, se obtuvieron 10 subsecuencias comunes a las Series de Tiempo de entrada. Para la Figura A.4  $FDTW = 434.00$ . Esta serie ha sido clasificada con un comportamiento complejo de origen natural [Bautista04]

En la Figura A.5 se muestra en color rojo la ST *Plasma* acotada a 120 puntos, en color azul la ST que corresponde a las 10 modificaciones que se realizaron sobre la ST *Plasma*, en este experimento se detectaron nueve subsecuencias similares. Para la Figura A.5  $FDTW = 1.5231$ .

En la Figura A.6 se muestra en color rojo la ST *White Noise* acotada a 200 puntos, en color azul la ST que corresponde a las 20 modificaciones que se realizaron sobre la ST *White Noise*, en este experimento se detectaron diecinueve subsecuencias similares. Para la Figura A.6  $FDTW = 7.386$ .



**Figura A.4.** Serie de Tiempo Láser (rojo), Serie de Tiempo Láser modificada en 5 puntos (azul), subsecuencias similares detectadas: diez



**Figura A.5.** Serie de Tiempo *Plasma* (rojo), Serie de Tiempo *Plasma modificada en 10 puntos* (azul), subsecuencias similares detectadas: nueve.



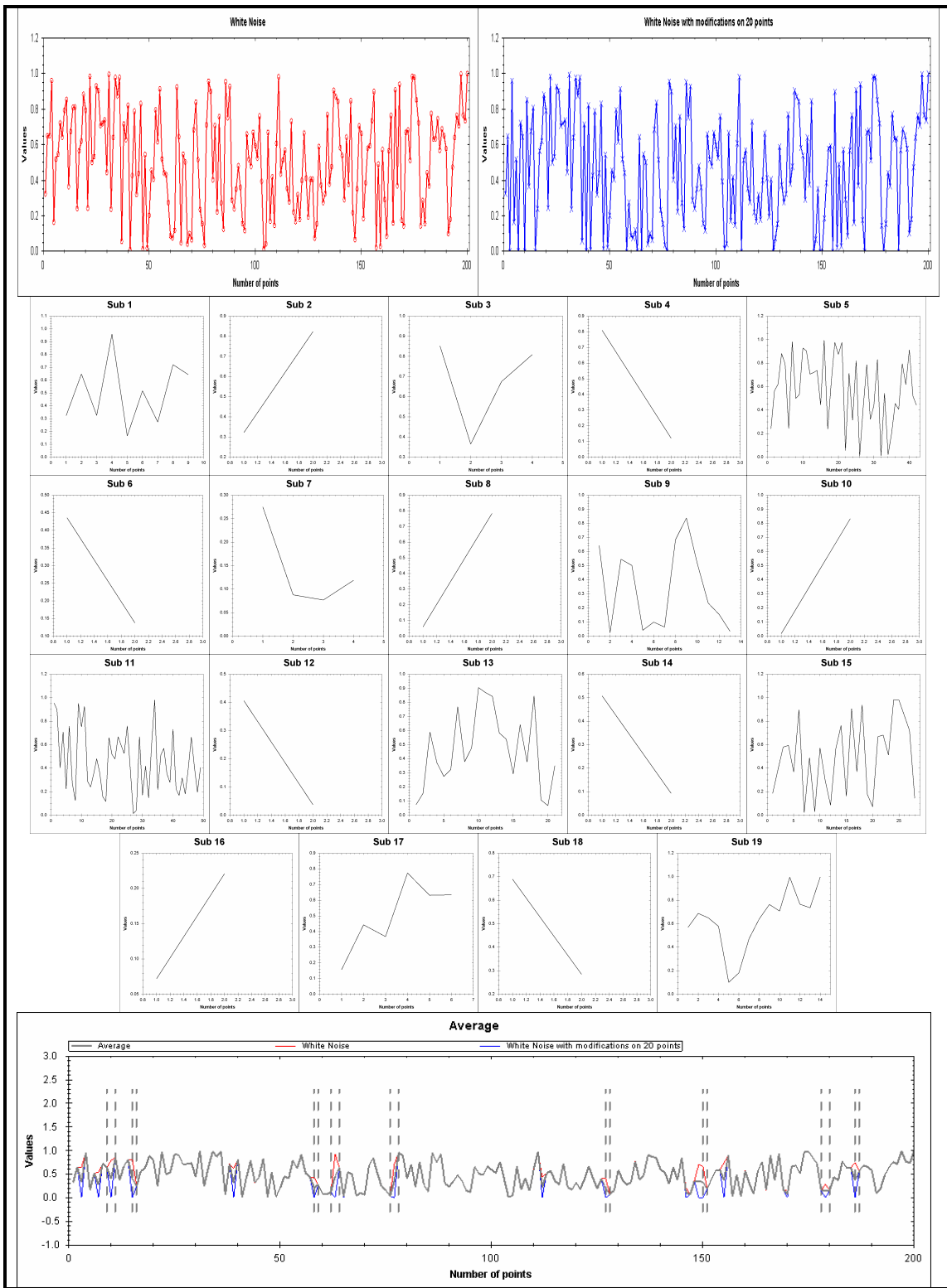


Figura A.6. Serie de Tiempo *White Noise* (rojo), Serie de Tiempo *White Noise modificada en 20 puntos* (azul), subsecuencias similares detectadas: 19.

# Creación Del Conjunto De Datos Con Series De Tiempo

A continuación se describe el conjunto de datos Series de Tiempo de origen Natural Artificial, mostradas en los experimentos al principio de este anexo. A este conjunto de datos, les fueron realizadas modificaciones sistemáticas en puntos aleatorios de las secuencias, tomando en cuenta los valores máximos y mínimos de la secuencia original. Por cada Serie de Tiempo de este conjunto de datos, se realizaron desde 1 hasta 20 modificaciones. Además, se realizaron pruebas experimentales sobre la base de datos “*Time Series Data Mining Archive*”, [TSDMA] para mostrar la utilidad del método propuesto.

1. **Seno** . Serie periódica generada para 10 ciclos, por la función:  $f(x) = \text{Seno}(x)$
2. **Vanderpol**. Serie periódica generada por una ecuación diferencial, que es modelo del paso de la carga eléctrica (denotada por  $y$ ) a través de un circuito oscilador de un tubo de vacío (triodo) .Su ecuación es:

$$\frac{d^2y}{dt^2} + (y^2 - \eta)\frac{dy}{dt} + \omega^2y = 0$$

3. **Qperiodic2**. Serie cuasi periódica obtenida de la medición de una variable de velocidad (cm/s) en un experimento anular para reproducir un Flujo de Couette (Reología), las mediciones se obtuvieron con un muestreo cada 0.1 segundos. Nota, las condiciones experimentales no se proporcionan en la literatura.

4. **Qperiodic3.** Serie cuasi periódica obtenida de la medición de una variable de velocidad (cm/s) en un experimento anular para reproducir un Flujo de Couette (Reología), las mediciones se obtuvieron con un muestreo cada 0.4 segundos. Nota, las condiciones experimentales no se proporcionan en la literatura.

5. **Mackey-Glass.** Serie caótica generada por una ecuación diferencial de retardo temporal: modelo de formación de células sanguíneas blancas (linfocitos). La ecuación es de la forma:

$$\frac{dx}{dt} = -bx(t) + \frac{ax(t - \tau)}{1 + [x(t - \tau)]^{10}}$$

donde  $a = 0.2$ ,  $b = 0.1$  y  $\tau = 3000$ .

6. **Cantor.** Serie caótica generada por el conjunto de Cantor (teoría de conjuntos), el cual es un conjunto cerrado que consiste enteramente de puntos de frontera cada uno de los cuales es un punto límite de dicho conjunto.

7. **Lorenz.** Serie caótica generada por un sistema de ecuaciones diferenciales: modelo de convección de fluidos (convección de Rayleigh-Benard) la cual se presenta en la atmósfera terrestre. El sistema de ecuaciones es de la forma:

$$\begin{aligned}\frac{dX}{dt} &= -\tilde{\sigma}X + \tilde{\sigma}Y \\ \frac{dY}{dt} &= -XZ + \tilde{r}X - Y \\ \frac{dZ}{dt} &= XY - \tilde{b}Z\end{aligned}$$

donde  $\sigma = 10$ ,  $r = 28$  y  $b = 8/3$  son parámetros adimensionales, X es proporcional a la velocidad del flujo de fluido circulatorio, Y caracteriza la diferencia de temperatura entre regiones de fluido ascendentes y descendentes y Z caracteriza la distorsión del perfil de temperatura vertical con respecto de su variación de equilibrio.

8. **Rosler.** Serie caótica generada por un sistema de ecuaciones diferenciales: modelo simplificado de Lorenz . El sistema de ecuaciones tiene la forma:

$$\begin{aligned}\dot{x} &= -(y + z) \\ \dot{y} &= x + 0.2y \\ \dot{z} &= 0.4 + xz - 5.7z\end{aligned}$$

9. **Laser** (Concurso Santa Fe). Serie compleja obtenida a partir de mediciones experimentales de la intensidad de pulsos de láser NH3 Infrarrojo Lejano, condiciones de la frecuencia:  $frecuencia\ serie\ Laser \geq 3 * frecuencia\ serie\ A1$ .

10. **Henon.** Serie caótica generada a partir de un mapa: modelo simplificado del mapa de Poincaré para el modelo de Lorenz. La expresión matemática es de la forma:

$$\begin{aligned}a &= 1.4; \\ b &= 0.3; \\ x(n+1) &= 1 - a * x^2(n) + y(n); \\ y(n+1) &= b * x(n); \end{aligned}$$

11. **EEG.** Serie compleja obtenida a partir de un electroencefalograma humano.

12. **Tent.** Serie caótica generada por un mapa de tipo lineal por partes. Su expresión matemática es de la forma:

$$x_{n+1} = 1 - 2 \left| x_n - \frac{1}{2} \right|$$

13. **Dow Jones.** Serie compleja obtenida a partir del Índice Industrial Dow Jones del NYSE (New York Stock Exchange), la serie corresponde al promedio semanal de los precios al cierre de operaciones para el periodo del 01/07/1900 al 03/08/1996.

**14. D1** (Concurso Santa Fe). Serie compleja generada por un modelo de la dinámica de una partícula amortiguada en un potencial de interacción. El potencial de Interacción es de la forma:

$$V = a_4(x_1^2 + x_2^2 + x_3^2 + x_4^2)^2 - a_2(x_1^2 * x_2^2)^{\frac{1}{2}} - a_1 * x_1$$

La fuerza se expresa como:

$$F = A * Sen(\omega * t)$$

en la dirección  $x_3$ , y la disipación es igual a:

Disipación =  $-\gamma * Velocidad$ . El valor de  $a_1$  tiene un pequeño desplazamiento producido por la integración de una variable aleatoria Gaussiana. El observable que se obtiene es:

$$\sqrt{(x_1 - 0.3)^2 + (x_2 - 0.3)^2 + x_3^2 + x_4^2}$$

**15. Kobe.** Serie compleja obtenida a partir de un acelerograma del sismo de Kobe del 16 de enero de 1995. La serie corresponde a la aceleración vertical, las mediciones fueron realizadas en Hobart, Australia por la Universidad de Tasmania, a partir de las 20:56:51 (GMT) durante 51 minutos a intervalos de 1 segundo.

**16. A1** (Concurso Santa Fe). Serie compleja obtenida a partir de mediciones experimentales de la intensidad de un láser NH3 Infrarrojo Lejano, las características del láser son las siguientes: láser de onda continua en el infrarrojo lejano, longitud de onda de 81.5 micras, el láser es bombeado ópticamente por un láser de N2O.

**17. ASCII TXT.** Serie compleja generada a partir de código de texto ASCII.

**18. El niño.** Serie compleja obtenida a partir de la medición experimental de la dinámica de una variable del fenómeno climático el niño.

**19. HIV DNA.** Serie compleja obtenida a partir del código del DNA del Virus de Inmunodeficiencia Humana HIV (1 = A, 2 = C, 3 = G, 4 = T).

**20. Human DNA.** Serie compleja obtenida a partir del código del DNA Humano.

**21. Lovaina** (Concurso Universidad de Lovaina). Serie compleja generada a partir del modelo generalizado del circuito de Chua. La expresión matemática es de la forma:

$$\begin{aligned}\dot{x}_1 &= \alpha [x_2 - h(x_1)] \\ \dot{x}_2 &= x_1 - x_2 + x_3 \\ \dot{x}_3 &= -\beta x_2\end{aligned}$$

Donde

$$h(x_1) = m_5 x_1 + \frac{1}{2} \sum_{i=1}^5 (m_{i-1} - m_i) (|x_1 + c_i| - |x_1 - c_i|)$$

y los valores de los parámetros son:  $\alpha = 9$ ,  $\beta = 14.286$  y de los vectores:

$m = [0.9/7, -3/7, 3.5/7, -2.7/7, 4/7, -2.4/7]$  y  $c = [1, 2.15, 3.6, 6.2, 9]$ .

**22. Plasma.** Serie compleja obtenida a partir de la medición de una variable de un experimento con plasma.

**23. SP500.** Serie compleja obtenida a partir del Índice Financiero de Standard & Pool para las 500 empresas mas importantes de la bolsa de valores de Nueva York.

**24. Star.** Serie compleja obtenida a partir de la medición de la intensidad luminosa de una estrella variable.

**25. Brownian Motion.** Serie estocástica generada a partir del modelado del movimiento browniano (proceso de ruido blanco integrado).

**26. White Noise.** Serie estocástica generada a partir del modelado de proceso de ruido blanco (ruido aleatorio uniforme).

**27. Logistic.** Serie caótica generada por un mapa: este mapa se puede pensar como un modelo ecológico de las variaciones anuales de poblaciones de insectos. Su expresión matemática es:

$$x_{n+1} = rx_n(1 - x_n)$$

donde  $n$  = año,  $x$  = Número de insectos que nacen,  $r$  = Número de huevos puestos por cada insecto en promedio que eclosionan al año  $n+1$ .

**28. Primos.** Serie compleja generada a partir de un conjunto de números primos.

**29. Ikeda.** Serie caótica generada a partir de un mapa construido en el plano complejo: modelo de la dinámica de pulsos de luz que viajan a través de un medio no lineal. La expresión matemática es de la forma:

$$z(n+1) = a + R * \exp(i(\phi - \frac{p}{(1 + |z^2(n)|)}))$$

donde  $z(n)$  representa al pulso que viaja a través de dicho medio. Los parámetros tienen los valores  $a = 1$ ,  $R = 0.9$ ,  $\phi = 0.4$  y  $p = 6$ .

**30. Random1.** Secuencia numérica de valores aleatorios. Se utilizó el generador de números Mersenne Twister [Matsumoto98].

## ANEXO D. Código Fuente

---

```
/*
Programa que identifica subsecuencias similares entre dos Series de
Tiempo, Miguel A. Mirón Bernal, CIC-IPN. 2007. Esta clase pertenece
al Proyecto FDTW_PR que pertenece a la Solución CodeSimilarity v2.5.
*/
using System;
using System.Collections.Generic;
using System.Text;
using System.IO;
using System.Collections;

namespace FDTW_PR
{
    public class node
    {
        //Items contained in FDTW - Path
        public int q, c; //index in Q and C Sequence
        public bool IsSub;
        public int nsub;
        public node() { IsSub = false; nsub = -1;} //Default Constructor
    }

    public class FDTW
    {
        //FDTW
        public double[] Q; //Sequence
        public double[] C; //Sequence
        public double[,] M; //Distance Matrix

        //FDTW & Pattern Recognition
        public node[] Path; //Array of index in M, for
subsequence detection
        public int patterns_founded;

        public int[] Qcode_line;
        public int[] Ccode_line;

        //Rest of variables
        public string FileQ, FileC; //Filenames for set the
sequences

        public FDTW() //Default Constructor
        { }

        //Set the input sequences from SeqQ,SeqC to Q,C respectively
        public void SetSequences(double[] _Q, double[] _C)
        {
            Q = (double[])_Q.Clone();
            C = (double[])_C.Clone();
        }
    }
}
```



```
//Returns de FDTW distance between Q,C Sequences
public double GetDistance()
{
    try
    {
        M = new double[Q.Length, C.Length];

        //First element
        M[0,0] = euclidean(Q[0],C[0]);

        //First Row
        for (int j = 1; j < C.Length; j++)
            M[0, j] = M[0, j - 1] + euclidean(Q[0], C[j]);

        //First Column
        for (int i = 1; i < Q.Length; i++)
            M[i, 0] = M[i - 1, 0] + euclidean(Q[i], C[0]);

        //Rest of table
        for (int i = 1; i < Q.Length; i++)
            for (int j = 1; j < C.Length; j++)
            {
                double distance = euclidean(Q[i], C[j]);

                M[i, j] = M[i - 1, j - 1] + distance;

                if ((M[i - 1, j] + distance) < M[i, j])
                    M[i, j] = M[i - 1, j] + distance;

                if ((M[i, j - 1] + distance) < M[i, j])
                    M[i, j] = M[i, j - 1] + distance;
            }

        //Gets the Warp path for subsequence detection
        this.BackTracking();

        //return de FDTW distance
        return M[Q.Length - 1, C.Length - 1];
    }

    catch (Exception ex)
    {
        Console.WriteLine(ex.ToString());

        //if an error occurs return Infinity
        return Double.MaxValue;
    }
}

//Returns the Euclidean distance between two values
private double euclidean(double q, double c)
{
    return (Math.Sqrt(Math.Pow(q - c, 2)));
}
```

```
//Performs BackTracking for further subsequence detection
private void BackTracking()
{
    double distance = 0.00;
    int i, j; //Distance Matrix index
    node p = new node(); //Contains one path_item

    ArrayList path = new ArrayList(); //Temporary Array

    p.q = i = Q.Length - 1; //Path always include the last
    p.c = j = C.Length - 1; //Matrix cell [Q,C]

    path.Add(p); //Add node to path

    //Move Along Distance Matrix until M[0,1] or M[1,0]
    while (i > 1 || j > 1)
    {
        p = new node();

        //Search the minimum distance from M[i,j] to
        //45,0, and 90 degrees neighbors

        //45 Degrees
        if (i >= 1 && j >= 1)
        {
            distance = M[i - 1, j - 1];
            p.q = i - 1;
            p.c = j - 1;
        }
        else
            distance = Double.MaxValue;

        //0 Grados
        if (i >= 0 && j > 0)
            if (M[i,j - 1] < distance)
            {
                p.q = i;
                p.c = j - 1;
                distance = M[i,j - 1];
            }

        //90 Grados
        if (i > 0 && j >= 0)
            if (M[i - 1,j] < distance)
            {
                p.q = i - 1;
                p.c = j;
            }

        i = p.q; //Move i,j index to closest neighbor with
        j = p.c; //minimum distance
        path.Add(p); //Add this node into path
    }
}
```

```

//Path always include the first Matrix cell M[0,0]
p = new node();
p.q = p.c = 0;
path.Add(p);

//Sort the elements for display
path.Reverse();
Path = (node[])path.ToArray(typeof(node));

int nsub = 0; bool found = false;

//Subsequence Detection
for (int z = 0; z < Path.Length - 1; z++)
    if (Math.Abs(Path[z].q - Path[z + 1].q) == 1 &&
        Math.Abs(Path[z].c - Path[z + 1].c) == 1)
    { //Do something with subsequence
        Path[z].IsSub = true;
        Path[z].nsub = nsub;
        Path[z + 1].IsSub = true;
        Path[z + 1].nsub = nsub;
        found = true;
    }
    else
        if (found)
        {
            nsub++; found = false;
        }
    patterns_founded = nsub + 1;
}

//Write the subsequences founded into a folder.
public void WriteSubsequences(string folder_dir,
    bool zeroAverage_oneSplit){
    try
    {
        string dir;
        if (zeroAverage_oneSplit)
            dir = folder_dir + "subs_" + ExtractFileName(FileQ) +
                "_" + ExtractFileName(FileC) + "\\";
        else
            dir = folder_dir + "AVsubs_" + ExtractFileName(FileQ)
                + "_" + ExtractFileName(FileC) + "\\";

        if (!Directory.Exists(dir))
            Directory.CreateDirectory(dir);

        //Split
        if (zeroAverage_oneSplit)
            for (int z = 0; z < Path.Length; z++)
            {
                node n;
                if (Path[z].IsSub)
                {
                    int sub = Path[z].nsub;

```

## ANEXO D. Código Fuente

---

```
using (StreamWriter sq = new StreamWriter(dir + (sub+1).ToString() +
    "_q.txt"))
{using (StreamWriter sc = new StreamWriter(dir + (sub+1).ToString()
    + "_c.txt"))
    {
        while (z < Path.Length && Path[z].nsub == sub)
        {
            n = Path[z];
            sq.WriteLine(Q[n.q].ToString());
            sc.WriteLine(C[n.c].ToString());
            z++;
        }
    }
    z--;
}
else
//Average
//Show Subsequences
for (int z = 0; z < Path.Length; z++)
{
    node n;
    if (Path[z].IsSub)
    {
        int sub = Path[z].nsub;
        using (StreamWriter sw = new StreamWriter(dir + (sub+1).ToString()
            + "_av.txt"))
        {
            while (z < Path.Length && Path[z].nsub == sub)
            {
                n = Path[z];
                sw.WriteLine(((Q[n.q] + C[n.c]) / 2));
                z++;
            }
        }
        z--;
    }
}
}
catch (Exception ex)
{
    Console.WriteLine(ex.ToString());
}
}
```

## ANEXO D. Código Fuente

---

```
/*
Proyecto que realiza la conversión del código fuente a secuencia
numérica. Miguel A. Mirón Bernal, CIC-IPN. 2007. Este proyecto
pertenece también a la Solución CodeSimilarity v2.5.
*/
using System;
using System.Collections.Generic;
using System.Text;
using System.IO;
using System.Collections;

namespace CSharpCodeConverter
{
    public class CodeConverter
    {
        //Nombre del codigo a convertir
        public string file;

        //Contiene en cada elemento del arreglo, una instruccion del
        //codigo fuente original
        private ArrayList code_lines = new ArrayList();

        public ArrayList converted_code = new ArrayList();
        public bool IsCategoryLevel;

        //Contendra la secuencia numerica que representa al codigo
fuente
        public double[] Q;

        //Contiene los valores de los operadores, y las categorias
de operadores
        public StartUp config;

        /* Remueve los comentarios del archivo fuente y divide en el codigo
        * en instrucciones, que se almacenan linea a linea en un ArrayList
        */
        private void RemoveComments()
        {
            using (StreamReader sr = new StreamReader(file))
            {
                while (!sr.EndOfStream)
                {
                    string line = sr.ReadLine();
                    int index;

                    if (line.Contains("//"))
                    {
                        //Es Comentario Simple
                        index = line.IndexOf("//");
                        line = line.Remove(index);
                    }
                }
            }
        }
    }
}
```

```

else if (line.Contains("/**"))
//Es Comentario Multiple
{
    index = line.IndexOf("/**");
    while (!line.Contains("*/"))
    {
        index = -1;
        line = sr.ReadLine();
    }
    int index2 = line.IndexOf("*/");
    if (index2 >= 0 && index != -1)
        line = line.Remove(index, (index2 + 2));
    else
        line = line.Remove(0, (index2 + 2));
}
//Dividir la lineas de codigo en instrucciones
string[] se = new string[] { "\t", "\n", "\r", ";" };
// string[] se = new string[] { "\t", "\n", "\r" };
string[] tokens = line.Split(se,
StringSplitOptions.RemoveEmptyEntries);
foreach (string token in tokens)
{
    if (token != "")
        code_lines.Add(token);
}
}
}

/* De las instrucciones almacenadas en code_lines, se remueven las
 * directivas using para codigos de C#.
 */
private void RemoveDirectives()
{
    ArrayList temp = new ArrayList();
    string t;

    for (int index = 0; index < code_lines.Count; index++)
    {
        t = (string)code_lines[index];
        if (t.Contains("using"))
            continue;
        else
            temp.Add(code_lines[index]);
    }
    code_lines.Clear();
    code_lines = (ArrayList)temp.Clone();
    temp.Clear();
}

/* Metodos principales, convierten las instrucciones en valores
numéricos, dependiendo del metodo que se utilice
TransformCodeOperatorLevel, ó TransformCodeCategoryLevel.*/

```

```
public void TransformCodeOperatorLevel()
{
    RemoveComments();
    RemoveDirectives();
    IsCategoryLevel = false;

    /*En el arreglo split deben encontrarse todos los
operadores del lenguaje
    * ademas de los separadores de clase y de funcion
    */
    string[] se = new string[] { "{", "}", " " };
    string[] split = new string[config.operators.Length +
se.Length];
    config.operators.CopyTo(split, 0);
    se.CopyTo(split, config.operators.Length);
    string line;
    for (int index = 0; index < code_lines.Count; index++)
    {
        line = (string)code_lines[index];
        string[] tokens = line.Split(split,
StringSplitOptions.RemoveEmptyEntries);
        string operaciones = line;

        //Diferencia del vector de tokens con la linea de codigo
        //para obtener las operaciones que afectan a esa linea de codigo
        for (int i = 0; i < tokens.Length; i++)
        {
            if (operaciones.Contains(tokens[i]))
            {
                int ind = operaciones.IndexOf(tokens[i]);
                operaciones = operaciones.Remove(ind,
tokens[i].Length);
                operaciones = operaciones.Insert(ind, "¿");
            }
        }
        ArrayList token_line = new ArrayList();
        int pos = 0;
        for (int j = 0; j < operaciones.Length; j++)
        {
            if (operaciones[j].Equals('¿'))
            {
                token_line.Add(TokenValue(tokens[pos++]));
            }
            else if (operaciones[j].ToString().Equals(" ")
                == false &&
                operaciones[j].ToString().Equals("{") ==
                false &&
                operaciones[j].ToString().Equals("}") ==
                false)
            {
                int next_quote = operaciones.IndexOf("¿", j);
                string op;
                if (next_quote > j)
                    op = operaciones.Substring(j, next_quote - j);
                else
```

```

        op = operaciones.Substring(j, operaciones.Length - j);
        token_line.Add(TokenValue(op));
        if (next_quote != -1)
            j = next_quote - 1;
        else
            break;
    }
}
//Aqui puede agregarse una estructura que lleve los tokens separados
y el numero de linea que le corresponde en el codigo original.
if (token_line.Count > 1)
{
    Code_Image ci = new Code_Image();
    ci.code_line = line;
    ci.code_img = token_line;
    converted_code.Add(ci);
}
}
ArrayList temp = new ArrayList();
if
(((Code_Image)converted_code[0]).code_img[0].GetType().Name.Equals("
Double"))
{
    foreach (Code_Image TkLn in converted_code)
    {
        foreach (double val in TkLn.code_img)
        {
            temp.Add(val);
        }
    }
    this.Q = (double[])temp.ToArray(typeof(double));
}
}

public void TransformCodeCategoryLevel()
{
    RemoveComments();
    RemoveDirectives();
    IsCategoryLevel = true;

    /*En el arreglo split deben encontrarse todos los
operadores del lenguaje
* ademas de los separadores de clase y de funcion
*/
    string[] se = new string[] { "{", "}", " " };
string[] split = new string[config.operators.Length + se.Length];
config.operators.CopyTo(split, 0);
se.CopyTo(split, config.operators.Length);
string line;
int openPh = 0;
for (int index = 0; index < code_lines.Count; index++)
{
    line = (string)code_lines[index];

```



```

        string[] tokens = line.Split(split,
StringSplitOptions.RemoveEmptyEntries);
        string operaciones = line;
        //Diferencia del vector de tokens con la linea de codigo
        //para obtener las operaciones que afectan a esa linea de codigo
        for (int i = 0; i < tokens.Length; i++)
        {
            if (operaciones.Contains(tokens[i]))
            {
                int ind = operaciones.IndexOf(tokens[i]);
                //operaciones = operaciones.Replace(tokens[i], "?");
                operaciones = operaciones.Remove(ind, tokens[i].Length);
                operaciones = operaciones.Insert(ind, "?");
            }
        }
        ArrayList token_line = new ArrayList();
        operaciones = operaciones.Trim();
        string[] ops = operaciones.Split(new Char[] { '?', ' '
' }, StringSplitOptions.RemoveEmptyEntries);
        foreach (string s1 in ops)
        {
            string s = s1.Trim();
            if (s.Contains("("))
            {
                openPh++;
                token_line.Add(CatValue("("));
                int ind = s.IndexOf("(");
                string s2;
                if (ind >= 0)
                {
                    s2 = s.Remove(ind, 1);
                    if(s2 != "")
                        token_line.Add(CatValue(s2));
                }
            }
            else if (s.Equals(""))
                openPh--;
            else if (s.Equals(")"))
                openPh--;
            else if (s.Contains(""))
            {
                token_line.Add(CatValue("")));
                int ind = s.IndexOf("");
                string s2;
                if (ind >= 0)
                {
                    s2 = s.Remove(ind, 1);
                    if (s2 != "" && s2 != "")
                        token_line.Add(CatValue(s2));
                }
            }
            else if (s.Equals("{") == false && s.Equals("}")
== false && s.Equals("") == false)
            {

```

```

        double v = CatValue(s);
        if (v != double.MaxValue)
            token_line.Add(CatValue(s));
        else
            Console.WriteLine("Error: TOKEN {0}
LINE: {1} ",s,line );
    }
}
//Aqui puede agregarse una estructura que lleve los tokens separados
y el numero de linea que le corresponde en el codigo original.
    if (token_line.Count > 0)
    {
        Code_Image ci = new Code_Image();
        ci.code_line = line;
        ci.code_img = token_line;
        converted_code.Add(ci);
    }
}
ArrayList temp = new ArrayList();
if
(((Code_Image)converted_code[0]).code_img[0].GetType().Name.Equals("
Double"))
{
    foreach (Code_Image TkLn in converted_code)
    {
        foreach (double val in TkLn.code_img)
        {
            temp.Add(val);
        }
    }
    this.Q = (double[])temp.ToArray(typeof(double));
}
temp.Clear();
}
/* Función que retorna el valor del token, es invocada desde
TransformCodeOperatorLevel*/
private double TokenValue(string _token)
{
    //Arreglo de palabras reservadas
    System.Collections.ArrayList reserved = config.reserved;
    //Arreglo de clases de operadores
    System.Collections.ArrayList categories =
config.categories;
    bool found = false;
    string token = _token.Trim();
    double value = double.MaxValue;
    //Buscar en las palabras reservadas
    foreach (item word in reserved)
    {
        if (token.CompareTo(word.id) == 0)
        {
            found = true;
            value = word.value;

```

```
        break;
    }
}

if (found)
    return value;
else
{
    //Buscar en los operadores
    foreach (TokenCategory cat in categories)
    {
        for (int i = 0; i < cat.objects.Count; i++)
        {
            item j = ((item)cat.objects[i]);

            if (token.CompareTo(j.id) == 0)
            {
                found = true;
                value = j.value;
                break;
            }
        }

        if (found) return value;
        else
        {
            //Realizar un cast, para determinar si es una constante
            try
            {
                double L = Convert.ToDouble(token);
                found = true;
            }
            catch (Exception) { found = false; }

            if (found) return //value = 0.7;
                value = config.constant_value;
            else
            // En caso contrario es una variable value = 0.9
                return
                    value = config.variable_value;
        }
    }
}

/* Función que retorna el valor de la categoria de un operador,
 * es invocada desde TransformCodeCategoryLevel
 */
private double CatValue(string _token)
{
    //Arreglo de clases de operadores
    System.Collections.ArrayList categories =
config.categories;
    string token = _token.Trim();
    double value = double.MaxValue;
}
```

```
//Buscar en los operadores
foreach (TokenCategory cat in categories)
{
    for (int i = 0; i < cat.objects.Count; i++)
    {
        item j = ((item)cat.objects[i]);

        if (token.CompareTo(j.id) == 0)
        {
            value = cat.value;
            break;
        }
    }
}
return value;
}

/* Una vez que se ha identificado un token, se busca en la
lista de palabras reservadas,
* en el arreglo de operadores, de no encontrarse, se
identifica como constante numerica
* o variable. Este metodo solo se utiliza para escribir a
un archivo el tipo de operador
* que le corresponde. Es invocado por WriteFileTransform().
*/

private string TokenType(double value)
{
    //Arreglo de palabras reservadas
    System.Collections.ArrayList reserved = config.reserved;

    //Arreglo de clases de operadores
    System.Collections.ArrayList categories =
config.categories;

    bool found = false;

    //Buscar en las palabras reservadas
    foreach (item word in reserved)
    {
        if (word.value == value)
        {
            found = true;
            break;
        }
    }

    if (found)
        return "RESERVED";
    else
    {
        //Buscar en los operadores
        foreach (TokenCategory cat in categories)
        {
```

```
        for (int i = 0; i < cat.objects.Count; i++)
        {
            item j = ((item)cat.objects[i]);

            if (j.value == value)
            {
                found = true;
                break;
            }
        }

        if (found) return "OPERATOR";
        else
        {
            if (value == config.constant_value)
                return "CONSTANT";
            else if (value == config.variable_value)
                // En caso contrario es una variable
                return "VARIABLE";
            else
                return null;
        }
    }
}

/* Funciones adicionales, necesarias solo para el manejo de
los datos de salida,
*/

}
/*Estructura que almacena la linea de codigo original, con
* sus valores correspondientes
*/
public struct Code_Image
{
    public ArrayList code_img;
    public string code_line;
}
}
```