Aplicaciones de Inteligencia Artificial

Research in Computing Science

Series Editorial Board

Research in Computing Science es una publicación trimestral, de circulación internacional, editada por el Centro de Investigación en Computación del IPN, para dar a conocer los avances de investigación científica y desarrollo tecnológico de la comunidad científica internacional. **Volumen 72,** mayo 2014. Tiraje: 500 ejemplares. *Certificado de Reserva de Derechos al Uso Exclusivo del Título* No. : 04-2005-121611550100-102, expedido por el Instituto Nacional de Derecho de Autor. *Certificado de Licitud de Título* No. 12897, *Certificado de licitud de Contenido* No. 10470, expedidos por la Comisión Calificadora de Publicaciones y Revistas Ilustradas. El contenido de los artículos es responsabilidad exclusiva de sus respectivos autores. Queda prohibida la reproducción total o parcial, por cualquier medio, sin el permiso expreso del editor, excepto para uso personal o de estudio haciendo cita explícita en la primera página de cada documento. Impreso en la Ciudad de México, en los Talleres Gráficos del IPN – Dirección de Publicaciones, Tres Guerras 27, Centro Histórico, México, D.F. Distribuida por el Centro de Investigación en Computación, Av. Juan de Dios Bátiz S/N, Esq. Av. Miguel Othón de Mendizábal, Col. Nueva Industrial Vallejo, C.P. 07738, México, D.F. Tel. 57 29 60 00, ext. 56571.

Editor responsable: *Grigori Sidorov, RFC SIGR651028L69*

Research in Computing Science is published by the Center for Computing Research of IPN. **Volume 72**, May 2014. Printing 500. The authors are responsible for the contents of their articles.

Printed in Mexico City, in the IPN Graphic Workshop – Publication Office.

Volume 72

Aplicaciones de Inteligencia Artificial

Obdulia Pichardo Lagunas (Ed.)

Instituto Politécnico Nacional, Centro de Investigación en Computación
México 2014

ISSN: 1870-4069

Instituto Politécnico Nacional (IPN)
Centro de Investigación en Computación (CIC)
Av. Juan de Dios Bátiz s/n esq. M. Othón de Mendizábal
Unidad Profesional "Adolfo López Mateos", Zacatenco
07738, México D.F., México

http://www.rcs.cic.ipn.mx
http://www.ipn.mx
http://www.cic.ipn.mx

Indexed in LATINDEX and Periodica / Indexada en LATINDEX y Periódica

Printing: 500 / Tiraje: 500

Printed in Mexico / Impreso en México

Prefacio

El propósito de este volumen es reflejar las nuevas direcciones de investigación y aplicaciones de los métodos de la Inteligencia Artificial.

Los artículos de este volumen fueron seleccionados con base en un estricto proceso de revisión efectuada por los miembros del Comité de revisión, tomando en cuenta la originalidad, aportación y calidad técnica de los mismos. Cada artículo fue revisado por lo menos por dos miembros del Comité de revisión del volumen.

Este volumen contiene 13 artículos relacionados con varios aspectos del desarrollo de los métodos de Inteligencia Artificial y ejemplos de sus aplicaciones a varias tareas tales como:

- detección de depredadores sexuales en redes sociales,
- clasificación automática de mensajes en Twitter,
- detección de fallas en motores eléctricos de potencia,
- agricultura,
- construcción de redes de ordenamiento,
- análisis de la calidad del aire,
- plegado de proteínas, entre otras.

Este volumen puede ser interesante para los investigadores y estudiantes de las ciencias de la computación, especialmente en áreas relacionadas con la inteligencia artificial y su aplicación a los diferentes ámbitos de la vida cotidiana; así como, para el público en general interesado en estos fascinantes temas.

En este número especial de la revista RCS, a nombre de la comunidad del programa de Ingeniería en Computación del CU UAEM Zumpango expresamos nuestro agradecimiento al Dr. Jorge Olvera García, Rector de la Universidad Autónoma del Estado de México (UAEM) y al M. en A. Rodolfo Téllez Cuevas, encargado del despacho de la dirección del Centro Universitario UAEM Zumpango, por apoyar de manera ingente la investigación, y el desarrollo de la ciencia y la tecnología, sustentado todo ello en un humanismo que transforma.

El proceso de revisión y selección de artículos se llevó a cabo usando el sistema libremente disponible EasyChair, www.EasyChair.org.

Obdulia Pichardo Lagunas
Mayo 2014

Table of Contents

Page

Detección de depredadores sexuales utilizando un sistema de consulta y clasificación supervisada

Yuridiana Alemán, Darnes Vilariño, David Pinto, Mireya Tovar

Facultad de Ciencias de la Computación, BUAP,
Puebla, Mexico

yuridiana.aleman@gmail.com, darnes@cs.buap.mx, dpinto@cs.buap.mx,
mireyatovar@gmail.com

Resumen. En este artículo se realiza un análisis entre técnicas de clasificación supervisada y el uso de un sistema de consultas como clasificador, dicho análisis se aplica a la detección de depredadores sexuales por medio de una metodología de dos fases. Primero, se reduce el corpus seleccionando aquellas conversaciones con más probabilidades de pertenecer a depredadores sexuales para posteriormente clasificar a los usuarios, finalmente se analizan los diálogos de los usuarios clasificados como depredadores. Los resultados obtenidos muestran que se obtienen mejores resultados con la clasificación supervisada, sin embargo, estos se incrementan cuando se fusionan ambas técnicas.

Palabras clave: Sistema de consulta, clasificación, *POStagger*, depredadores.

1. Introducción

Gracias al avance de la tecnología, las redes sociales han avanzado a pasos agigantados y con ello, la forma en que los depredadores sexuales hacen contacto con una víctima. El FBI cataloga a estos adultos como "viajeros", porque van de un lugar a otro para tener relaciones sexuales con niños que conocieron por Internet. Estas personas acechan las salas de chat, buscando convencer a los adolescentes de participar en coqueteos cibernéticos. Con el paso del tiempo, estos adultos desarrollan romances a distancia, para después intentar persuadir a sus víctimas a entablar relaciones íntimas.

Dada esta situación, se han hecho varios avances en la detección automática de depredadores sexuales, empresas como *Facebook* han impulsado este tipo de investigaciones, sin embargo, todavía quedan muchos aspectos que tomar en cuenta para que la detección sea eficaz. Algo que obstruye este tipo de investigaciones es la escasez de material para el análisis, además, se debe tomar en cuenta otros aspectos como el tipo de escritura usado en los chats, así como el hecho de que existen conversaciones que contienen términos obscenos, pero no es propiamente el caso de un depredador sexual, y viceversa, es decir, los depredadores no siempre usan términos de ese tipo, si no un lenguaje mas formal.

Dados los datos disponibles, en el presente estudio se analizarán únicamente conversaciones en el idioma inglés.

En este artículo se realizan diversos experimentos para la detección de conversaciones en donde participan depredadores sexuales e identificar cual de los usuarios de dicha conversación es que el intenta persuadir a los demás. La propuesta consta de 2 etapas, primeramente se realiza una clasificación de conversaciones, para posteriormente detectar los usuarios que son depredadores sexuales. Para ambas fases se utilizan métodos de clasificación supervisada con diferentes conjuntos de características, el sistema de consultas únicamente se utiliza en la primera fase.

El artículo está estructurado de la siguiente manera. En la sección II se detalla es estado del arte referente a este tema de investigación. La sección III muestra la metodología planteada y el preprocesamiento dado a las conversaciones. La sección IV muestra los resultados obtenidos en las fases 1 y 2 respectivamente. Finalmente, la sección V muestra las conclusiones obtenidas y el trabajo futuro para esta investigación.

2. Estado del arte

La mayoría de los trabajos que abordan la temática de detección de depredadores sexuales toman como punto de referencia la investigación presentada en [8], donde se realiza un estudio piloto sobre el uso de técnicas de clasificación automática de textos para identificar depredadores sexuales en línea. Se trabaja con un corpus obtenido del sitio *Perverted Justice*[1]. Este estudio se presenta como una investigación inicial para la detección del *grooming attack*[2]. En los experimentos realizados, se identifican los textos de los depredadores sexuales y los textos de las víctimas (clasificación en dos clases) utilizando Máquinas de soporte vectorial (*SVM*) y *k-vecinos mas cercanos* (*k-NN*). Se hicieron varias pruebas con diferentes características (de 5,000 a 10,000), obteniendo mejores resultados con un conjunto de 10,000 características con *k-NN* (k=30).

En los últimos años se han publicado varias investigaciones sobre esta temática, la mayoría provenientes del congreso *CLEF PAN Lab on Uncovering Plagiarism, Authorship, and Social Software Misuse*[3]. En dicho congreso se abordan temas relacionados a la atribución de autoría, plagio de textos y vandalismo informático. A partir del año 2012 se incluyó dentro de atribución de autoría la subtarea *"Sexual Predator Identification"*.

El *F-score* mas alto en la subtarea fue alcanzado por [12], en esta investigación no se realiza ningún tipo de preprocesamiento, sólo un filtrado para quitar aquellas conversaciones muy cortas, con caracteres no entendibles o donde los

[1] *http://www.perverted-justice.com/*

[2] Definido en [4] como "proceso de comunicación por el cual un autor aplica estrategias de búsqueda de afinidad, mientras que simultáneamente adquiere información sobre sus víctimas con el fin de desarrollar las relaciones que resulten en cumplimiento de su necesidad, por ejemplo acoso sexual físico"

[3] *http://pan.webis.de/*

participantes tienen pocas intervenciones, con esto se logra reducir drásticamente la dimensión del corpus. Además, proponen una metodología de dos etapas, una para clasificar las conversaciones donde interviene al menos un depredador sexual y otro basado en los diálogos por persona para distinguir a los depredadores de las víctimas o pseudovíctimas. Para la primera clasificación se obtuvo mayor precisión con SVM y *tf-idf*, para el caso del segundo modelo se obtienen mejores resultados con redes neuronales.

Todas las investigaciones mencionadas, tienen en común que utilizan técnicas de clasificación para resolver la tarea, sin embargo, en [1] se propusieron dos metodologías diferentes. Una de ellas está basada en técnicas de recuperación de información, para lo cual se desarrolló un diccionario de términos sexuales con 919 entradas. En cada una de estas entradas se encuentran un conjunto de términos similares a un sentido en particular. Las conversaciones fueron representadas con la técnica de *Posting List*[5]. Cada entrada del diccionario sexual se considera una consulta y se devuelven las primeras 10 conversaciones que de alguna manera incluyen términos alusivos al sexo. Esta propuesta logró detectar mayor número de depredadores, pero también reportó un gran número de conversaciones que no necesariamente están asociadas a individuos que buscan un favor sexual, sino que utilizan términos obscenos para comunicarse.

3. Metodología

Para la realización de los experimentos se utiliza como *training* las conversaciones extraídas del sitio web de la fundación *PJFI.org* (Estados Unidos), la cual pretende contrarrestar a los pedófilos que intentan acercarse a una víctima por medio de la red. En el sitio web se publican conversaciones de depredadores sexuales con sus "víctimas", que en realidad son personas que se hacen pasar por adolescentes y/o niños. Además, se añade al conjunto el *training* de la competencia "PAN 2012", obtenidas del sitio web de dicha competencia (*http://pan.webis.de/*). El *test* de dicha competencia se utiliza también como *test* de los experimentos realizados. El comité organizador proporciona el *gold* de los usuarios etiquetados como depredadores.

En base a los conjuntos que datos con los que se cuentan, se propone una metodología como se muestra en la figura 1. En general, la propuesta se divide en dos fases principales:

1. Clasificación de las conversaciones en dos posibles tipos: *"Predator"* y *"No Predator"*.
2. Clasificación de los diálogos que conforman las conversaciones recuperadas en la fase 1 para detectar si un usuario es una víctima y un depredador.

3.1. Preprocesamiento de corpus

Una vez determinados los conjuntos de conversaciones para el *training* y *test*, se obtuvieron algunos promedios y medidas para su análisis, los cuales se

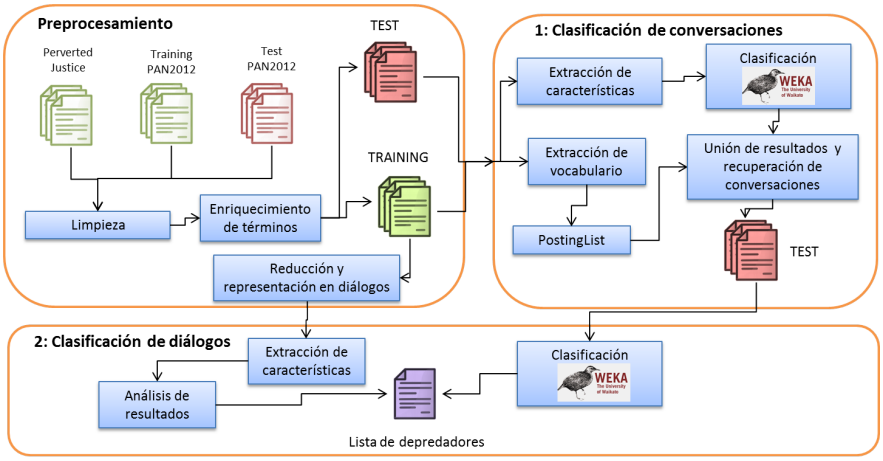


Fig. 1. Metodología propuesta para la investigación.

muestran en la tabla 1[4]. Aquí se observa que el corpus de entrenamiento tiene más riqueza en cuanto a la extensión de las conversaciones, aunque cabe señalar que los valores son muy extremos, por ejemplo en el *training* la conversación con mayor número de usuarios es de 30, mientras que en el *test* es de 115. Para el caso de los usuarios, el que participa en mas conversaciones tiene 30 en el *training*, mientras que en el *test* la mayor participación es de 128 conversaciones.

Tabla 1. Descripción del corpus usado.

DATO	*Training*	*Test*
Vocabulario	317,450	624,755
Conversaciones de Depredadores	2,353	3,715
Conversaciones de No Depredadores	64,884	151,413
Depredadores	480	250
No depredadores	97,807	218,431
Usuarios por conversación	2.28	2.29
Conversaciones por usuario	1.62	1.56
Líneas	17.24	13.23
Longitud	568.30	532.24
Palabras	108.85	96.16

Algo notable es que el vocabulario del *test* es considerablemente mayor al vocabulario del *training*, pero dada la naturaleza de los textos, mas que palabras diferentes, pueden existir distintos modos de escritura para una sola palabra, por ejemplo, en salas de chat es muy común poner el término "lol", pero si un usuario escribe "lol" y "lool", se tomarán como dos palabras distintas.

Las conversaciones contienen un exceso de términos no reconocidos por un diccionario, además de que abundan los emoticones, cadenas de símbolos no

[4] Los últimos cinco datos están expresados en promedios

reconocibles, que pueden ser URLs, imágenes, entre otros, ésto en gran medida por el tipo de vocabulario utilizado en estas conversaciones, además, algunas abordan temas de programación, por lo que el número de símbolos extraños aumenta. Por lo tanto, se construyeron 3 recursos léxicos (diccionarios) para ayudar a enriquecer los textos. Estos recursos son:

1. **Emoticones**[5]: Se obtuvo una lista con los emoticones mas comunes, dicha lista fue enriquecida con los emoticones predefinidos de *Facebook* y *Gmail*. La lista recopilada cuenta con 344 elementos.
2. **Contracciones**[6]: Esta lista contiene alrededor de 65 contracciones más usadas en los Estados Unidos.
3. **Vocabulario SMS**: Es una lista obtenida de [10], la cual contiene 820 abreviaciones o simplificaciones más usadas en SMS y chats, donde el tiempo de respuesta es importante y generalmente no se toman en cuenta reglas ortográficas y gramaticales.

Tanto en el *training* como en el *test*, todas las ocurrencias de alguno de estos recursos fueron sustituidas por su correspondiente significado, además de documentar el número de ocurrencias de cada tipo por conversación para futuros experimentos. Con los cambios hechos al corpus, el vocabulario disminuyó significativamente (de 317,455 a 181,291 palabras para el caso del *training* y de 624,755 a 360,880 palabras para el *test*).

3.2. Conjuntos de Características

En experimentos preliminares se trabajó con varios conjuntos de características, pero en algunos los resultados eran muy bajos o simplemente inviables de procesar dada la magnitud del corpus, como el caso del uso del vocabulario. Por lo tanto, se escogieron las características que se considera pueden diferenciar los textos de una víctima, de los textos del depredador. Los conjuntos seleccionados se mencionan a continuación.

SUBDUE: Se analiza el *training* con la herramienta *SUBDUE*[7], a fin de extraer las palabras mas representativas del texto utilizando la extracción de sub-estructuras mas comunes con una representación de grafos. Se utiliza una representación basada en la notación *gSpan*[13], en la cual un grafo es representado como una 4-tupla $G = (V, E, L, l)$ donde V es un conjunto (no vacio) de vértices, E es un conjunto de aristas de la forma $E \subseteq V \times V$, L es un conjunto de etiquetas y l es una función que asigna una etiqueta a un par de vértices asociados. En la figura 2 se muestra un ejemplo de esta representación con el texto "of course i need help with the machine".

Cabe mencionar que el tiempo de ejecución para extraer esta representación depende de la cantidad de palabras pertenecientes a cada conjunto, por lo que es inviable aplicarlo en el *training* de la fase 1.

[5] *http://www.netlingo.com/smileys.php*
[6] "Corrupciones fonéticas", en *http://es.wikibooks.org*

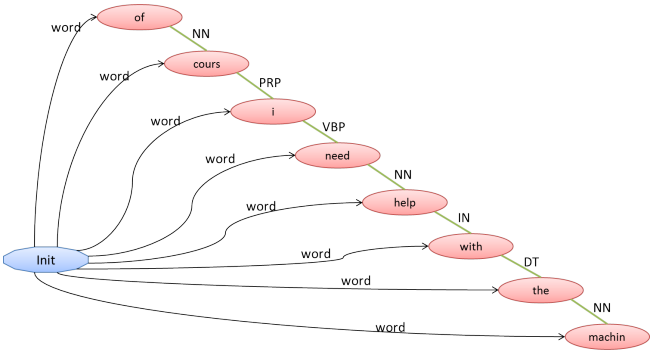


Fig. 2. Ejemplo de representación basada en secuencias de palabras.

Características Léxicas: Se realiza un conteo de algunos símbolos y palabras escritas de una manera singular. En la tabla 2 se listan dichas características, las cuales son obtenidas en la fase de preprocesamiento. Cabe mencionar que en este conjunto están contemplados los conteos de cada elemento de la lista, por lo que todos los atributos son numéricos, excepto el de la clase.

Tabla 2. Características léxicas.

Palabras que inician con mayúscula	URLs	"?"
Emoticones	Palabras escritas en mayúsculas	"."
Palabras truncadas	Contracciones	"!"
Números	";"	Total de signos utilizados
","	":"	

Categorías Gramaticales: Se utilizó la herramienta *POS-tagger* de la Universidad de Stanford[11] para obtener las partes del discurso y el lema de cada término dentro de las conversaciones. Al igual que en el conjunto anterior, fueron considerados como características los conteos de las apariciones por conversación.

Sufijos: Se tomaron como atributos los sufijos existentes para el idioma inglés donde cada sufijo representa un atributo y las veces que aparece en cada conversación es el valor para dicho atributo. Un sufijo se define como "Un afijo que va pospuesto y, en particular, los pronombres que se juntan al verbo y forman con él una sola palabra"[7].

Signos: Se realiza el conteo de todos los signos existentes en *string.punctuation*[8], la cual es una constante definida en el módulo *String* de *python*. Esta constante contiene una cadena con los signos de puntuación en *ASCII* (figura 3).

[7] `http://www.rae.es/`
[8] https://docs.python.org/2/library/string.html

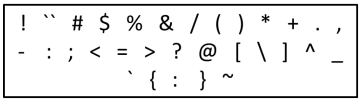


Fig. 3. Cadena de símbolos contenida en *string.punctuation*.

3.3. *Posting List*

Partiendo de la idea de un sistema básico de consultas, se creó un algoritmo que clasifica las conversaciones del *test* en base a un *Posting List*[5] construido con las conversaciones del *training* (Algoritmo 1).

Algoritmo 1 Algoritmo para consultas con *Posting List* basado en trigramas de lemas.

Entrada: $CONVER$: Conjunto de conversaciones del *test* sin palabras cerradas.
Entrada: $INDICE$: Elementos del índice.
Entrada: $LIST$: Lista de conversaciones del *training* con su respectiva categoría.
Entrada: $TOPE$: Porcentaje de documentos tomados en cuenta para la asignación de la categoría.
Salida: $CATEGORIA$: Lista de conversaciones con una categoría asignada.

para $a \in Conver$ **hacer**
  $lemas \leftarrow Extrae\ Lemas\ Conversacion(a, Conver)$
  para $b \in lemas$ **hacer**
    $post \leftarrow Extrae\ Claves\ Conversacion(b, INDICE)$
    para $c \in post$ **hacer**
      $Valor \leftarrow Extrae\ Valor\ Trigrama(c, INDICE)$
      $peso_c \leftarrow peso_c + Valor$
      $conteo_c++$
    fin para
  fin para
  para $d \in conteo$ **hacer**
    si $conteo_d >= TOPE$ **entonces**
      $etiqueta \leftarrow Extrae\ Categoria(d, LIST)$
      $Categoria_{etiqueta} \leftarrow Categoria_{etiqueta} + peso_d$
    fin si
  fin para
  si $Categoria_{\text{“depredador”}} <= 0$ **and** $Categoria_{\text{“no predator”}} <= 0$ **entonces**
    $CATEGORIA_a \leftarrow$ “*sin palabras*”
  si no, si $Categoria_{\text{“depredador”}} == Categoria_{\text{“no predator”}}$ **entonces**
    $CATEGORIA_a \leftarrow$ “*sin categoria*”
  si no, si $Categoria_{\text{“depredador”}} > Categoria_{\text{“no predator”}}$ **entonces**
    $CATEGORIA_a \leftarrow$ “*predator*”
  si no
    $CATEGORIA_a \leftarrow$ “*no predator*”
  fin si
fin para
retornar $CATEGORIA$

Para la creación de dicho recurso, se extraen los lemas de cada palabra y se eliminan las palabras cerradas, posteriormente se crean trigramas de lemas para anexarlos a un índice, en donde se incluye el trigrama y la lista de conversaciones donde este aparece. Además, para cada trigrama se calcula su *tf-idf*, el cual le asigna un peso para cada trigrama dentro de la colección de conversaciones. Posteriormente se tomaron en cuenta las siguientes consideraciones:

- Algunos términos de las conversaciones del *test* no se encuentran en el índice creado, por lo que estos no se consideran en la consulta.
- En la asignación de la categoría, se utiliza una variable llamada TOPE, la cual determina el porcentaje de documentos necesarios para la asignación de la categoría final. Por lo tanto, una conversación del *test* es tomada en cuenta si el número de términos en los que aparece es mayor o igual a dicha variable.
- Se presentan casos de conversaciones en donde ningún término supera el TOPE, o en su defecto, existen el mismo número de documentos positivos y negativos para asignar la categoría, por lo tanto, se toman en cuenta las siguientes categorías para asignar a una conversación:
  - **"Predator"**: Cuando hay más conversaciones positivas que negativas que contengan los términos de la consulta.
  - **"No-predator"**: Cuando hay más conversaciones negativas que positivas que contengan los términos de la consulta.
  - **"Sin palabras"**: La conversación no tiene palabras que estén incluidas en el índice, o la frecuencia de las conversaciones retornadas no superan el TOPE.
  - **"Sin categoría"**: Se tiene la misma frecuencia de conversaciones en las dos categorías.

3.4. Métricas de evaluación

Los resultados de los diferentes experimentos fueron evaluados con las siguientes métricas utilizadas en la recuperación de información.

1. **Precisión (P):** Fracción de los datos recuperados que son relevantes.

$$P = \frac{items\ relevantes\ recuperados}{items\ recuperados} \tag{1}$$

2. **Recuerdo (R):** Fracción de los datos relevantes que son recuperados.

$$R = \frac{items\ relevantes\ recuperados}{items\ relevantes} \tag{2}$$

3. ***F-score*:** Media armónica entre la precisión y el recuerdo.

$$F - score = 2 * \frac{P * R}{P + R} \tag{3}$$

4. Resultados obtenidos

Para realizar los experimentos de clasificación supervisada se utilizaron varios algoritmos implementados en la herramienta *WEKA*[3], después de experimentos preliminares y dado el tipo de conjuntos de características utilizados, se decidió utilizar árbol de decisión[9], redes neuronales[6] y bosque aleatorio[2]. Además, se implementa un sistema de voto, el cual consiste en tomar como positiva una

conversación si por lo menos uno de los tres clasificadores utilizados la clasificó como tal. Se utilizaron los conjuntos individuales y las uniones de tales conjuntos. En la tabla 3 se muestran los resultados obtenidos.

En la tabla se muestran las conversaciones de depredadores que fueron recuperadas (Rec Pos), precisión y recuerdo, además, para homogeneizar los datos se agrega la variable *F-score* y el número de depredadores incluidos en las conversaciones recuperadas (No Dep). Se observa que los mejores resultados en todas las métricas utilizadas se presentan en el sistema de voto, utilizando diferentes combinaciones de características. La precisión mas alta fue alcanzada utilizando solamente el conjunto de categorías gramaticales utilizando el algoritmo de redes neuronales, sin embargo, dado que ésta se considera una primera fase, se prefiere que el número de depredadores encontrados sea alto para poder proseguir con el análisis de clasificación de la segunda etapa de la propuesta inicial. Por otro lado, aunque el mejor *F-score* fue obtenido utilizando los conjuntos de signos y categorías, se obtienen mas conversaciones con depredadores utilizando los conjuntos de características léxicas y categorías gramaticales, esto se debe a que en el *test* existen usuarios que participan en mas de una conversación, por lo que un mayor *f-score* no implica una mayor cantidad de depredadores recuperados

Tabla 3. Resultados para clasificación con los conjuntos de características.

Conjunto	**Clasificador**	**Rec Pos**	**Precisión**	**Recuerdo**	***F-score***	**No Dep**
Léxicas	Árbol de decisión	1,057	0.306	0.285	0.295	210
	Bosque aleatorio	1,247	0.444	0.336	0.382	214
	Redes neuronales	1,182	0.288	0.318	0.302	211
	Voto	1,431	0.217	0.385	0.277	221
Categorías	Árbol de decisión	923	0.568	0.248	0.346	195
	Bosque aleatorio	1,012	0.627	0.272	0.380	207
	Redes neuronales	851	**0.719**	0.230	0.348	190
	Voto	1,249	0.481	0.336	0.396	216
Sufijos	Árbol de decisión	494	0.554	0.133	0.214	155
	Bosque aleatorio	476	0.563	0.128	0.209	160
	Redes neuronales	277	0.632	0.075	0.133	113
	Voto	709	0.459	0.191	0.270	186
Signos	Árbol de decisión	476	0.452	0.128	0.200	139
	Bosque aleatorio	508	0.342	0.137	0.195	153
	Redes neuronales	156	0.378	0.042	0.076	80
	Voto	714	0.304	0.192	0.236	172
Léxicas categorías	Árbol de decisión	1,148	0.421	0.309	0.356	213
	Bosque aleatorio	1,204	0.602	0.324	0.421	216
	Redes neuronales	1,192	0.488	0.321	0.387	213
	Voto	1,588	0.337	**0.427**	0.377	**226**
Signos categorías	Árbol de decisión	1,056	0.494	0.284	0.361	202
	Bosque aleatorio	1,015	0.698	0.273	0.393	207
	Redes neuronales	980	0.758	0.264	0.391	199
	Voto	**1,492**	0.483	0.402	**0.439**	225
Sufijos signos	Árbol de decisión	831	0.493	0.224	0.308	181
	Bosque aleatorio	818	0.590	0.220	0.322	190
	Redes neuronales	43	0.589	0.012	0.023	35
	Voto	1,104	0.452	0.297	0.359	202
Léxicas sufijos	Árbol de decisión	1,058	0.463	0.285	0.353	210
	Bosque aleatorio	1,063	0.584	0.286	0.384	201
	Redes neuronales	1,183	0.486	0.318	0.385	215
	Voto	1,454	0.342	0.391	0.365	221

En cuanto al sistema de recuperación de información, se realizaron varios experimentos con distintos valores de la variable TOPE, los resultados se muestran en la tabla 4. Se puede observar que a medida que la variable TOPE se reduce, se incrementan las conversaciones recuperadas y por lo tanto, las métricas evaluadas. Además, se recuperan más conversaciones que en los experimentos de clasificación supervisada, aunque con un *F-score* menor.

Tabla 4. Resultados para la clasificación de conversaciones utilizando el sistema de consultas.

TOPE	Sin palabras	Sin categoría	Rec Pos	*Precision*	*Recall*	*F-score*	No Dep
100	112,054	239	709	0.274	0.191	0.225	192
90	110,983	239	833	0.306	0.224	0.259	201
80	103,868	241	852	0.303	0.230	0.261	202
70	97,220	242	878	0.300	0.236	0.264	204
60	85,707	232	953	0.280	0.256	0.267	211
50	71,454	333	1,142	0.265	0.307	0.285	224
40	64,311	334	1,350	0.266	0.363	0.307	228
30	52,874	336	1,717	0.269	0.462	0.340	236
20	42,463	332	2,079	0.277	0.560	0.371	239
10	38,634	330	2,107	**0.319**	**0.567**	**0.408**	**240**

4.1. Clasificación de usuarios

Para la generación del *training* de la segunda etapa se utiliza el mismo conjunto inicial de conversaciones, pero eliminando aquellas que son muy cortas, que contienen muchos caracteres no imprimibles y en donde sólo exista un usuario.

Se obtuvo un conjunto de 15,374 conversaciones, las cuales se dividieron en dos categorías de diálogos: "predator" y "no-predator". Además, se aplicó el preprocesamiento y la expansión de términos utilizados en el conjunto inicial de conversaciones obteniendo un *training* de 26,472 diálogos.

Para la creación del *test*, se crea un conjunto según las conversaciones recuperadas en la fase uno. Se fusionaron los 2 conjuntos de conversaciones que contienen mas depredadores sexuales (240 y 226 depredadores), es decir las conversaciones recuperadas con el sistema de voto utilizando el conjunto de características "Léxicas Categorías" y las conversaciones recuperadas del sistema de consultas con un "TOPE" igual a 10.

Como se puede observar, ninguno de los conjuntos citados contiene los 250 depredadores del *test* original, ya que con la unión de los conjuntos sólo se llegó a 246 depredadores, por lo que se analizaron de manera manual todas las conversaciones donde participan los 6 depredadores faltantes. De manera general, se observó que éstos depredadores participan en mas de una conversación, sin embargo, son los únicos usuarios dentro de cada conversación y los diálogos que escriben son muy cortos y con demasiados emoticones y URLs. Dadas estas características, las conversaciones de estos usuarios no se asemejan a las conversaciones de depredadores incluidas en el *training*, donde generalmente hay mas diálogos y por lo menos son dos usuarios por conversación.

Para esta etapa, los resultados del sistema de consulta fueron demasiado bajos, por lo que se omiten en el análisis, sin embargo, el tamaño del *training* es adecuado para extraer las subestructuras utilizando la herramienta de *SUBDUE*. Se utilizaron las mismas combinaciones de características con los mismos clasificadores de la fase 1, pero ahora para los nuevos conjuntos resultantes de dicha fase. En la gráfica 4 se muestra solamente el *F-score* de todos los experimentos realizados, se omiten las otras métricas por que al ser la última etapa, se necesita recuperar a los usuarios que realmente sean depredadorese sexuales, pero sin que tengan falsos positivos, esto se logra analizando los resultados de esta métrica.

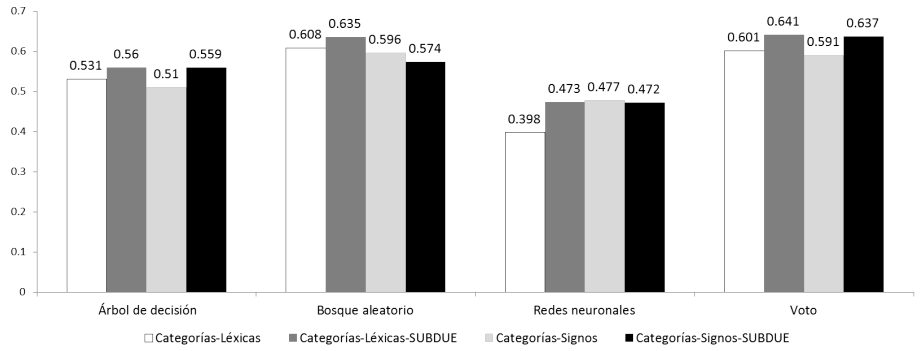


Fig. 4. *F-Score* para los conjuntos de características utilizados en la fase uno.

El *F-score* mas alto se obtiene con el algoritmo de bosque aleatorio utilizando los conjuntos de características léxicas, categorías gramaticales y la herramienta *SUBDUE*. Además, se observa en la gráfica que el clasificador de redes neuronales generó el *F-score* mas bajo para todos los conjuntos de características, incluso al combinarlas con *SUBDUE*. Algo interesante es que los resultados no siguen el mismo patrón que en la fase anterior, donde uno de los mejores *F-score* es reportado con el uso de redes neuronales, en este caso, se puede afirmar que en esta fase dicho algoritmo entorpece los resultados finales..

El sistema de voto demuestra que tiene mejor exactitud a la hora de detectar a los depredadores sexuales de las víctimas, incluso con el ruido que puede tener el uso de redes neuronales como parte del sistema. Al final de los experimentos, se logra un *F-score* de 64.1 % en la detección de los depredadores sexuales utilizando los conjuntos de características léxicas, conteos de categorías gramaticales y las palabras extraídas con la herramienta *SUBDUE*.

5. Conclusiones y trabajo futuro

De los experimentos reportados en esta investigación, se puede concluir lo siguiente:

- El *training* y especialmente el *test* tienen demasiados elementos propios de chats. Con la utilización de los tres diccionarios creados se logra extender los textos y agregarle significado a algunas palabras que no lo tienen, esto hace que el vocabulario se reduzca considerablemente y que los etiquetadores utilizados en fases posteriores tengan un mejor comportamiento. Sin embargo, actualmente los chats utilizan un conjunto mucho mayor de símbolos, palabras e incluso imágenes, las cuales se insertan en la conversación tecleando un código numérico (esto ocurre especialmente en *facebook*), por lo que a pesar de tener una metodología para el preprocesamiento de los datos, esta no logra enriquecer todos los términos utilizados en las conversaciones actuales.
- En la primera fase se experimenta con varios conjuntos de características y técnicas de recuperación de información. Los resultados sirven como un filtro para las conversaciones, de tal manera que en la fase de clasificación de usuarios exista un balance entre las conversaciones de depredadores y las conversaciones de otras personas. Las complicaciones en esta fase radican en la magnitud del corpus, así como en el desbalanceo de las clases. Además, se observa que los conjuntos de características utilizados en esta etapa de la investigación no fueron suficientes para obtener buenos resultados en la segunda etapa.
- En la fase de clasificación de usuarios, el *F-score* obtenido significa un incremento notable respecto a los resultados obtenidos anteriormente para esta misma tarea, sin embargo, aún no supera los resultados obtenidos a nivel mundial.
- A pesar de que los resultados no superan el estado del arte mundial, se considera que la metodología propuesta aporta una nueva visión para tratar esta problemática y otras similares como la detección del género (donde ya se han logrado resultados aceptables con los mismos conjuntos de características) y tareas que involucren textos de redes sociales.
- El sistema de consulta ofrece la oportunidad de clasificar las conversaciones tomando en cuenta el vocabulario usado, además, necesita menos tiempo y recurso computacional para ejecutarse, aunque sus resultados con muy bajos, logra reducir drásticamente el conjunto de conversaciones, manteniendo las que son positivas, es decir, en las que participa por lo menos un depredador sexual.

Como trabajo futuro se pretende incluir los siguientes aspectos:

- Incrementar los diccionarios creados a fin de reforzar el significado de los textos. Se considera agregar mas elementos como emoticonos compuestos, tomar en cuenta la repetición de las letras (por ejemplo que las palabras "hello" y "hellloo" sean consideradas iguales), analizar errores gramaticales, sinónimos y búsqueda de analogías en los diálogos. Esto ayudará también a la reducción de vocabulario y por consecuencia, a la reducción en los tiempos de creación de modelos y clasificación.
- La extracción de nuevas características semánticas que no hayan sido consideradas en este trabajo tomando en cuenta otros aspectos de la conversación

como la hora, tiempo de respuesta, relación entre los conceptos utilizados y las fechas en que se dieron dichas conversaciones.
- Expandir el análisis de los textos mediante un estudio utilizando el cambio de parámetros de los clasificadores utilizados a fin de encontrar la combinación óptima para el incremento del *f-score*.

Además se está trabajando en un estudio comparativo utilizando la misma metodología aquí presentada en otros corpus a fin de estudiar el comportamiento de los conjuntos de características utilizados en cuanto a las métricas de precisión y *F-score*.

Referencias

1. Ayala, D.V., Castillo, E., Pinto, D., Olmos, I., León, S.: Information retrieval and classification based approaches for the sexual predator identification. In: Forner, P., Karlgren, J., Womser-Hacker, C. (eds.) CLEF (Online Working Notes/Labs/Workshop) (2012)
2. Breiman, L.: Random forests. Mach. Learn. 45(1), 5–32 (Oct 2001)
3. Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., Witten, I.H.: The weka data mining software: an update. SIGKDD Explor. Newsl. 11(1), 10–18 (Nov 2009)
4. Harms, C.M.: Grooming: An operational definition and coding scheme 8(1), 1–6 (2007)
5. Manning, C.D., Schütze, H.: Foundations of statistical natural language processing. MIT Press, Cambridge, MA, USA (1999)
6. Mitchell, T.M.: Machine Learning. McGraw-Hill, Inc., New York, NY, USA, 1 edn. (1997)
7. Olmos, I., Gonzalez, J.A., Osorio, M.: Subgraph isomorphism detection using a code based representation. In: Russell, I., Markov, Z. (eds.) FLAIRS Conference. pp. 474–479. AAAI Press (2005)
8. Pendar, N.: Toward spotting the pedophile telling victim from predator in text chats. In: Proceedings of the International Conference on Semantic Computing. pp. 235–241. ICSC '07, IEEE Computer Society, Washington, DC, USA (2007)
9. Quinlan, J.R.: C4.5: Programs for Machine Learning (Morgan Kaufmann Series in Machine Learning). Morgan Kaufmann, 1 edn. (Oct 1992)
10. Symens, B.: Acronyms Dictionary for Texting Chatting E-mail. Rebecca J Symens
11. Toutanova, K., Klein, D., Manning, C., Singer, Y.: Feature-rich part-of-speech tagging with a cyclic dependency network. In: Human Language Technology Conference (HLT-NAACL 2003) (2003)
12. Villatoro-Tello, E., Juárez-González, A., Escalante, H.J., Montes-y-Gómez, M., Pineda, L.V.: A two-step approach for effective detection of misbehaving users in chats. In: CLEF (Online Working Notes/Labs/Workshop) (2012)
13. Yan, X., Han, J.: gspan: Graph-based substructure pattern mining. In: ICDM. pp. 721–724 (2002)

Laboratorio en línea para el procesamiento automático de documentos

Julio C. Torres López, Christian Sánchez-Sánchez, Esaú Villatoro-Tello

Departamento de Tecnologías de la Información,
División de Ciencias de la Comunicación y Diseño,
Universidad Autónoma Metropolitana, Unidad Cuajimalpa, D.F., México

210368282@alumnos.cua.uam.mx, {csanchez, evillatoro}@correo.cua.uam.mx

Resumen. Las grandes cantidades de información textual que actualmente se generan y almacenan digitalmente, junto con la dificultad que existe para analizarla, hace necesario el desarrollo de herramientas que faciliten este trabajo. Existen diferentes campos en las Ciencias de Computación y la Lingüística que en conjunto posibilitan el desarrollo de este tipo de herramientas; en particular una de estas áreas del conocimiento es el Procesamiento de Lenguaje Natural (PLN). El PLN investiga y formula mecanismos computacionalmente efectivos que facilitan la interacción hombre-máquina permitiendo una comunicación mucho más fluida y menos rígida que los lenguajes formales. Sin embargo, para usuarios poco experimentados en este campo, asimilar este tipo de procesos no es algo trivial, situación que desmotiva al uso de las mismas. Con la finalidad de apoyar el desarrollo y la investigación en áreas afines al PLN, en este artículo se presenta un Laboratorio Virtual en Línea para el Procesamiento Automático de Documentos desarrollado en la Universidad, donde se puedan realizar experimentos y ver resultados de forma inmediata, en diferentes tareas relacionadas con el procesamiento automático del lenguaje.

Palabras clave: Preprocesamiento, normalización, etiquetado POS, entidades nombradas, análisis sintáctico, clasificación de textos.

1. Introducción

El avance en la tecnología al día de hoy, así como su bajo costo, ha fomentado que cualquier persona, organismo o empresa pueda almacenar de forma digital grandes cantidades de información textual. El análisis que se pueda realizar a esta información puede ayudar a obtener una mejor perspectiva su contenido y así aportar más y mejores elementos durante los procesos de toma de decisiones.

Dada esta situación, contar con herramientas que ayudan a realizar un procesamiento automático o semi-automático de la información, en particular texto; en grandes volúmenes, ayuda a ahorrar tiempo y recursos.

Existen diferentes campos en las Ciencias de Computación y la Lingüística que en conjunto posibilitan el desarrollo de este tipo de herramientas; en particular una de estos áreas es el Procesamiento de Lenguaje Natural (PLN).

El Procesamiento de Lenguaje Natural o PLN, es una rama de la Inteligencia Artificial, que dentro de sus objetivos tiene el habilitar a las computadoras a procesar y "entender" el texto. El PLN investiga y formula mecanismos computacionalmente efectivos que facilitan la interacción hombre-máquina y permiten una comunicación mucho más fluida y menos rígida que los lenguajes formales, facilitando, teóricamente, la comprensión o análisis de grandes cantidades de información digital. Así entonces, el PLN propone varias técnicas que ayudan a procesar, clasificar y entender (hasta cierto punto) grandes volúmenes de información obtenida [1].

El Procesamiento de Lenguaje Natural representa un área de investigación muy variada. Entre algunos de los temas más representativos se tiene como campos de investigación importantes a: la extracción de información, la generación automática de resúmenes, la búsqueda de respuestas, la recuperación de información monolingüe y multilingüe, técnicas automáticas de clasificación de textos temática y no temática, identificación de perfiles de usuarios, análisis de sentimientos, etc., mostrando grandes avances en todas ellas [2]. Es importante mencionar que a pesar de los avances logrados, el PLN sigue aún en desarrollo, y la necesidad de cada vez más y mejores herramientas motiva el desarrollo de nuevas técnicas y/o enfoques que contribuyan en esta área del conocimiento cada vez más útil en los diferentes sectores tanto sociales como industriales.

En ese sentido, dentro de este trabajo se describe el desarrollo realizado para el laboratorio en línea, especializado en diversas tareas involucradas en el procesamiento automático de documentos, el cual tiene como principal objetivo convertirse en un sitio de referencia y de gran utilidad para estudiantes, desarrolladores e investigadores involucrados en el área de la Lingüística Computacional.

El resto del documento se encuentra organizado de la siguiente manera: en la sección 2 se presenta el marco teórico, más relevante, relacionado a la temática en cuestión y algunas de las herramientas existentes. En la sección 3 se describe el sistema propuesto así como su arquitectura. Finalmente, la sección 4 muestra las conclusiones obtenidas y define las líneas de trabajo futuro.

2. Trabajo relacionado

2.1. Marco teórico

Dentro de esta sección se describen algunas de las principales tareas que se utilizan de manera regular en muchos de los problemas que aborda el área del PLN (Véase Sección 1).

Preprocesamiento: La etapa de preprocesamiento de los documentos consiste fundamentalmente en preparar los documentos para su análisis, eliminando aquellos elementos que se consideran no necesarios para algunas tareas. El preprocesamiento tiene procesos tales como:

a) Eliminación de los términos o partes del documento que no son objeto de indexación, por ejemplo la eliminación de palabras funcionales[1] (preposiciones, artículos, etc.), las etiquetas XML o cabeceras de los documentos.
b) La normalización de textos, consiste en homogeneizar todo el texto de una colección de documentos sobre la que se trabajará, y que afecta, por ejemplo, a la consideración de los términos en mayúscula o minúscula; el control de determinados parámetros como cantidades numéricas o fechas; el control de abreviaturas y acrónimos [3].
c) La lematización es un proceso que busca encontrar la raíz léxica de las palabras, en este sentido las múltiples derivaciones y/o inflexiones de una palabra son llevadas a una sola, el morfema original. Cuando un proceso de lematización no es posible de realizar, se recurre a un proceso de truncado, cuya finalidad es aproximar lo más posible las palabras a su raíz léxica.

En general, el preprocesamiento de los documentos tiene como objetivo principal facilitar el manejo de los documentos en etapas posteriores, dado que ayuda a reducir el costo requerido tanto en espacio como en tiempo de cómputo al momento de procesarlos.

Etiquetado POS: El etiquetado POS[2], también conocido como etiquetado de parte de la oración es el proceso de asignar (o etiquetar) a cada una de las palabras de un texto su categoría gramatical (sustantivos, verbos, artículos, etc.). Este proceso se puede realizar de acuerdo con la definición de la palabra o considerando el contexto en que aparece dentro de algún texto.

El etiquetado gramatical muestra, hasta cierto punto, la estructura de un documento, brinda una gran cantidad de información sobre una palabra y sus vecinas. Una etiqueta gramatical puede ofrecer información relacionada con la pronunciación, el reconocimiento de sustantivos, adjetivos, tipo de derivación y/o inflexión.

Dependiendo del tipo de problema que se quiera resolver, el contar con etiquetas POS dentro de un documento puede resultar de gran utilidad.

Identificadores de Entidades Nombradas (NER): Permite identificar en un texto personas, organizaciones, lugares o fechas.

El NER puede ser utilizado por sistemas que necesiten conocer de quiénes se está hablando y en qué momento para extraer información de los textos que procesan. Es muy útil en sistemas cuya funcionalidad consiste en filtrar información no deseada, o en sistemas que requieren llevar información no estructurada a un formato estructurado, así por ejemplo, construir una base de datos a partir de información identificada por un reconocedor NER [12]. Otro ejemplo puede ser aquel que filtra contenido para adultos, con la finalidad de que dicha información no sea mostrada en las aulas de una escuela de educación básica.

[1] A las palabras funcionales también se les conoce como palabras cerradas o *stopwords*.
[2] Por sus siglas en inglés, *Part-Of-Speech*

Analizador Sintáctico o *Parse tree*: Es una forma ordenada de descomponer una oración según su forma gramatical y estructura. Encontrando las relaciones entra cada una de las palabras que conforman la oración. Teniendo en cuenta que una gramática enlista los principios bajo los cuales se agrupan las palabras, es el conjunto de reglas que describe que es válido en un lenguaje.

Clasificación de Textos (CT): El objetivo de la clasificación de textos es colocar, de forma automática, documentos dentro de un número fijo de categorías (temas o clases) predefinidas, en función de su contenido[13].

En su forma más simple, el problema de clasificación de textos puede formularse de la siguiente manera: Dados un conjunto de documentos de entrenamiento $\mathcal{D}_{Tr} = \{d_1, \ldots, d_n\}$ y un conjunto de categorías predefinidas $\mathcal{C} = \{c_1, \ldots, c_m\}$, el objetivo de la CT es formular la función de aprendizaje que sea capaz de generar el modelo de clasificación adecuado (*i.e.,* una hipótesis) $h : \mathcal{D} \rightarrow \mathcal{C}$ la cual será capaz de clasificar de manera correcta todos aquellos documentos no vistos durante el entrenamiento que pertenecen a $\mathcal{D}$.

Es importante mencionar que en la primera versión de este proyecto, *i.e.,* el Laboratorio en Línea para el Procesamiento de Automático de Documentos, se han considerado, desarrollado e implementado varias herramientas que impactan de manera directa en la resolución de las tareas mencionadas en esta sección.

2.2. Herramientas existentes

En la actualidad existen muchas herramientas de apoyo a tareas relacionadas con el PLN, sin embargo, en la mayoría de los casos estas herramientas cuentan con interfaces gráficas complicadas de entender para no expertos en el tema, o simplemente no se cuentan con ningún tipo de interfaces. Por otro lado, normalmente requieren de realizar un proceso de instalación de diferentes componentes, los cuales en muchos casos, generan gran variedad de dificultades para poder utilizar las herramientas debido a las múltiples incompatibilidades que hay entre versiones de compiladores y/o incluso de sistemas operativos. Agregado a esto, muchas de estas herramientas carecen de manuales de usuario comprensibles para usuarios nuevos en la temática; así mismo, no permiten la integración de más herramientas. Todo esto resulta en un problema que desmotiva su uso y provoca que en la mayoría de los casos, tanto estudiantes como investigadores opten por desarrollar desde cero diferentes módulos y/o herramientas que les permitan realizar sus experimentos.

A continuación se muestra un análisis y comparación de algunas herramientas utilizadas en el PLN[3].

Weka. Es una extensa colección de algoritmos de aprendizaje desarrollados por la universidad de Waikato (Nueva Zelanda) implementada en Java. Son útiles para ser aplicados sobre datos mediante las interfaces que ofrece el sistema o para embeberlos dentro de cualquier aplicación. Además Weka contiene las

[3] Las herramientas analizadas se seleccionaron debido a su popularidad entre varios grupos de investigación.

herramientas necesarias para realizar aprendizaje sobre los datos, tareas de clasificación, agrupamiento y visualización. Weka está diseñado como una herramienta orientada a la extensibilidad por lo que añadir nuevas funcionalidades es teóricamente posible.

La licencia de Weka es GPL, lo que significa que este programa es de libre distribución y difusión. Además, ya que Weka está programado en Java, es independiente de la arquitectura, pues funciona en cualquier plataforma sobre la que haya una máquina virtual Java disponible. Sin embargo, y pese a todas las cualidades que Weka posee, tiene un gran defecto, y éste es la escasa documentación orientada al usuario poco experimentado y que junto a una usabilidad bastante pobre, lo convierte en una herramienta difícil de comprender y manejar si no se cuenta con información adicional. Weka es una herramienta muy especializada, y requiere que los usuarios tengan conocimientos amplios sobre técnicas de aprendizaje automático para darle un uso apropiado y por consecuencia poder explotarla correctamente [4][5].

Natural Language Toolkit (NLTK). NLTK proporciona un conjunto de bibliotecas de procesamiento de textos para la clasificación, simbolización, derivado, etiquetado, análisis sintáctico y semántico-razonamiento. NLTK es adecuado para los lingüistas, ingenieros, estudiantes, educadores, investigadores y usuarios de la industria por igual. NLTK está disponible para Windows, Mac OS X y Linux. NLTK es una librería de código abierto, está desarrollado en Python [6]. Es una programa muy poderoso para el PLN ya que contiene un gran conjunto de herramientas, sin embargo, si la persona que lo va a utilizar no tiene conocimientos previos en Phyton, esto representará un gran esfuerzo para poder utilizarlo de manera eficaz, el tiempo que tomará aprendiendo todas sus funcionalidades podría manejarse como una desventaja, además de que todo se realiza desde línea de comandos. .

Text to Matrix Generator (TMG). Text to Matrix Generator (TMG) es un toolbox que se puede utilizar para diversas tareas en la minería de texto. La mayor parte de TMG (versión 6.0; Dic. '11) está escrito en MATLAB, aunque una gran parte de la fase de indexación de la versión actual del TMG está escrito en Perl. TMG es especialmente adecuado para aplicaciones de minería de datos. Originalmente construido como una herramienta de procesamiento previo para la creación de matrices término-documento (TDM) de texto no estructurado, la nueva versión de TMG (Diciembre '11) ofrece una amplia gama de herramientas para las siguientes tareas: indexación, recuperación, reducción de dimensionalidad, agrupamiento y clasificación. El proceso de indexación puede incluir varios pasos, como la eliminación de las palabras cerradas, como artículos y conjunciones, la eliminación de términos muy frecuentes o infrecuentes. TMG acepta como archivos de entrada, archivos en texto ASCII y muchos archivos PostScript y PDF. TMG permite como opción, una variedad de sistemas de expresión de ponderación y normalización. Una de las mayores desventajas de este toolbox es que a pesar de ser de uso libre, MATLAB no lo es [7].

Stanford CoreNLP. Proporciona un conjunto de herramientas de PLN, para documentos de texto los cuales pueden estar en diferentes idiomas. A partir de

texto plano, se pueden ejecutar todas las herramientas programando las líneas de código correspondientes. Stanford CoreNLP contiene las herramientas de PLN: etiquetador de categorías gramaticales (POS), identificación de entidades nombradas (NER), el analizador, y el sistema de resolución de la correferencia, y proporciona los archivos de modelos para el análisis de inglés y algunos otros idiomas. El objetivo de este conjunto de herramientas de Stanford es permitir a usuarios la obtención, de forma rápida y sin problemas de anotaciones lingüísticas completas de textos en lenguaje natural. Está diseñado para ser altamente flexible y extensible y con la opción de que se pueden seleccionar las herramientas que deben estar habilitadas o inhabilitadas. El código CoreNLP Stanford está escrito en Java y bajo la Licencia Pública General de GNU (v2 o posterior) [8]. Una desventaja grande de este conjunto de herramientas es también que si no existen conocimientos previos en programación en lenguaje Java, será difícil su utilización. Otra desventaja es que no existe una visualización gráfica de los resultados de la misma y todo debe manejarse desde línea de comandos.

Estos conjuntos de herramientas son en su mayoría de uso gratuito, algunas ya son ampliamente conocidas y otras más están en proceso de crecimiento. Algunas funcionan con diferentes idiomas, pero el idioma predominante es el inglés. En importante recalcar la variedad de lenguajes de programación empleados (*e.g.*, JAVA, C, Python, etc.) hace difícil la interacción entre todas ellas. Finalmente, es conveniente mencionar que existen una gran diversidad de herramientas de análisis que son ajenas a estos grandes sistemas, pero que pueden ser de utilidad. Ejemplos de estas son herramientas de análisis en redes sociales como Twitter o Facebook [9].

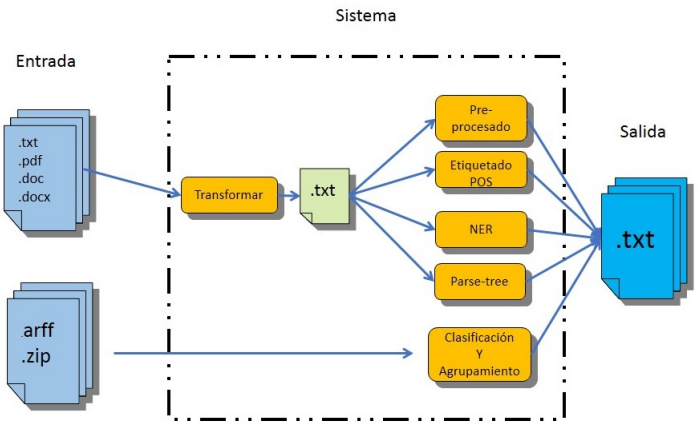


Fig. 1. Arquitectura del sistema propuesto.

3. Sistema propuesto

El sistema propuesto para este proyecto cuenta con características que buscan facilitar el uso de sus herramientas del PLN . Teniendo en cuenta que el laboratorio fue pensado para ser utilizado por estudiantes con poca o ninguna experiencia en el procesamiento de documentos su diseño se simplificó lo más posible. El laboratorio cuenta con diferentes módulos los cuales integran una serie de herramientas para las tareas más utilizadas en el PLN.

Como se muestra en la Figura 1 el sistema tiene una arquitectura modular. El sistema acepta documentos de entrada en diferentes formatos. Estos documentos al ingresarlos al sistema son trasformados a formato de texto plano (.txt). Este archivo podrá utilizar los módulos de preprocesado, etiquetado POS, Identificados de entidades nombradas (NER) y *parse tree*. También acepta archivos de tipo ARFF (.arff), formato exclusivos para poder utilizar la herramienta WEKA, a estos no se le realiza ninguna transformación y se pueden utilizar directamente en tareas de clasificación. También es posible agregar archivos comprimidos ZIP (.zip) que cuenten con una estructura de árbol donde existe una carpeta raíz que almacena más carpetas y estos a su vez otros documentos. Con los archivos comprimidos el sistema puede construir archivos arff, utilizando los nombres de las carpetas como el atributo clase y sus contenidos como sus atributos y/o características.

3.1. Descripción de los módulos

En esta sección realizaremos una descripción de cada uno de los módulos que utilizamos en nuestro sistema.

a) El módulo de transformación de documentos se realiza la modificación de los archivos originales los cuales pueden ser de tipo *.txt, .pdf, .doc* y *.docx*, este módulo los transforma a formato *.txt*, esto se realiza para poder tener un manejo más simple de los archivos. Los archivos *.txt* son guardados en una carpeta que tendrá el nombre del usuario, los archivos con los formatos originales no son guardados por nuestro sistema, para ahorrar espacio en nuestro servidor (Figura 2).

b) El módulo de preprocesado de documentos se realizan las diferentes tareas para preparar los documentos de una manera más rápida y sencilla. En este módulo se utilizaron algunas herramientas de los diferentes programas descritos previamente (Véase Sección 2) y se desarrollaron algunos más (eliminar números, puntos, acentos). Solo basta con seleccionar un archivo que ya previamente se ha cargado al sistema y se elige que preprocesamiento se va a realizar (Figura 3).

Los diferentes preprocesados que realiza el módulo son los siguientes:

- Palabras cerradas. Elimina en el documento las palabras que no tienen relevancia semántica en nuestros documentos como pueden ser preposiciones, determinantes, pronombres y conjunciones del idioma inglés.

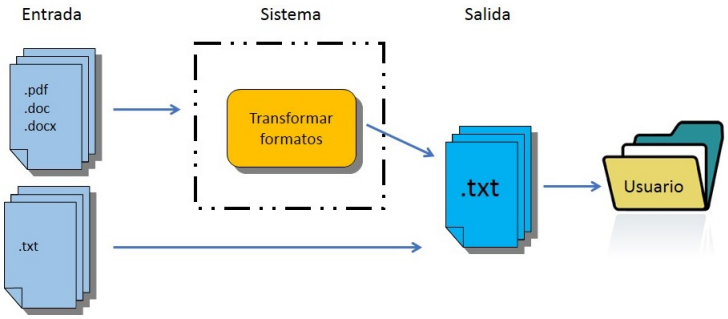


Fig. 2. Módulo de transformación.

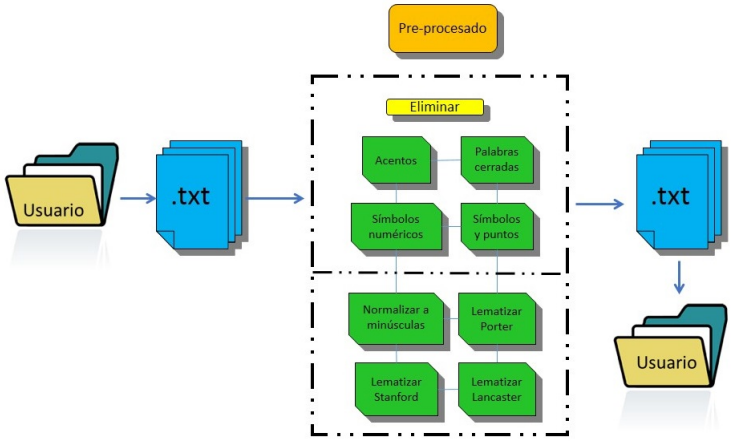


Fig. 3. Módulo de preprocesado.

- Símbolos y puntos. Elimina del documento los símbolos de puntuación y símbolos como: ¡¿ % $, etc.
- Acentos. Elimina en el document los diferentes símbolos de acentuación como tilde, diéresis, etc.
- Números. Elimina en el document todos los símbolos numéricos.
- Normalizar. Hace que nuestro documento sea transformado todo a letra minúscula.
- Lematizador. Transforman cada palabra de nuestro texto a su lema utilizando la herramienta que proporciona NLTK con el algoritmo de Porter y Lancaster [6] respectivamente. También se puede utilizar el Lematizador desarrollado por el Grupo de Procesamiento de Lenguaje Natural de Stanford [8].

La salida de este módulo es un archivo *.txt* con el resultado del preprocesado seleccionado, agregando el nombre del proceso al nombre original del archivo

tratado. Se pueden aplicar las diferentes tareas a un mismo archivo en el orden que se prefiera. El archivo resultante se guarda en la carpeta del usuario, es importante mencionar que el archivo original continúa existiendo sin sufrir ningún cambio.

c) El módulo de etiquetado POS, realiza un etiquetado al archivo original agregando su categoría gramatical (sustantivos, verbos, artículos, etc.) a cada palabra de nuestros archivos. En este módulo está compuesto por herramientas que realizan el etiquetado a todo tipo de archivos y una más que está especializada en documentos de Twitter, ya que reconoce algunos emoticones y anotaciones características usados en esta red social (Figura 4).

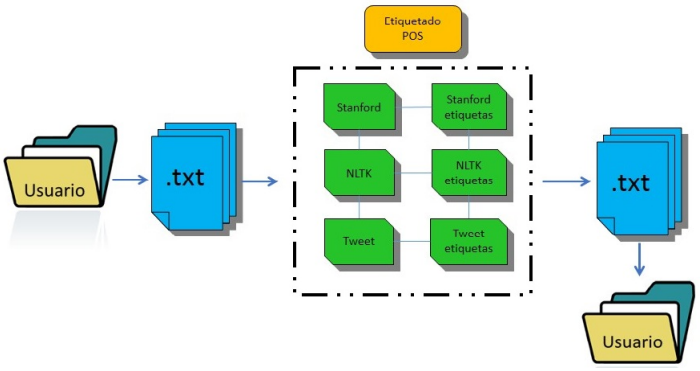


Fig. 4. Módulo de Etiquetado POS.

Las herramientas de etiquetado POS son las proporcionadas por el Grupo de Procesamiento de Lenguaje Natural de Stanford [8] y NLTK [6], de estas herramientas podemos escoger dos modalidades; la primera es que la etiqueta (categoría gramatical) aparezca al lado de la palabra asociada y la segunda es que solo aparezcan las etiquetas. La herramienta especializada en Twitter fue desarrollada por el grupo del Instituto de Tecnologías del Lenguaje, Escuela de Ciencias de la Computación de la Universidad Carnegie Mellon [9].

d) El módulo de reconocimiento de entidades nombradas (NER), reconoce diferentes entidades las cuales pueden ser nombres propios, organizaciones, fechas y lugares. En este módulo se utilizan dos herramientas diferentes, una proporcionada por el Grupo de Procesamiento de Lenguaje Natural de Stanford [8] y la otra de LingPipe [10]. Con la herramienta de Stanford se puede visualizar como salida las entidades junto con los términos o ver solamente las entidades. Con la herramienta de LingPipe solo se visualizan las etiquetas correspondientes al texto (Figura 5).

Este módulo funciona como los anteriores donde se puede seleccionar un archivo precargado en el sistema y aplicarle los diferentes procesos, en el cual se le anexará al final del nombre del archivo el proceso que se le realizó.

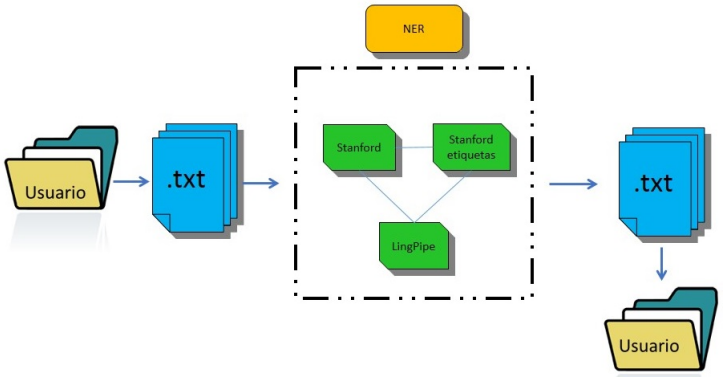


Fig. 5. Módulo de Reconocimiento de Entidades Nombradas.

e) El módulo de *Parse Tree*, descompone una oración según sus funciones gramaticales y nos genera un archivo con éstas por cada oración. Como algunas veces es difícil de leer este tipo de archivo, por ejemplo cuando el texto es muy grande. Para facilitar la visualización se integrá la herramienta llamada DEPENDENSEE [11] que a través de una imagen nos permite ver de una forma gráfica el árbol sintáctico generado. Guarda en una imagen (.png) de cada oración que se tenga en el texto, estas imágenes se guardan en las carpetas personales, y al finalizar la sesión son eliminadas (Figura 6).

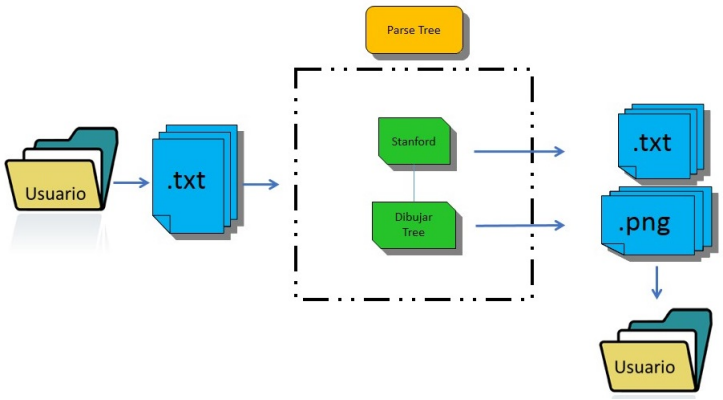


Fig. 6. Módulo de *Parse Tree*.

f) El módulo de clasificación, emplea principalmente módulos de la herramienta WEKA. Debido a esto, es necesario que los documentos a clasificar estén en un archivo *.arff*. Sin embargo, el resultado de este módulo es en un archivo de texto que muestra la matriz de confusión, el porcentaje de casos clasificados

correctamente, el porcentaje de precisión (*precision*), recuerdo (*recall*) y medida F (*f-measure*). De momento solo se pueden utilizar dos algoritmos de clasificación que son los J48 y Naïve Bayes (Figura 7).

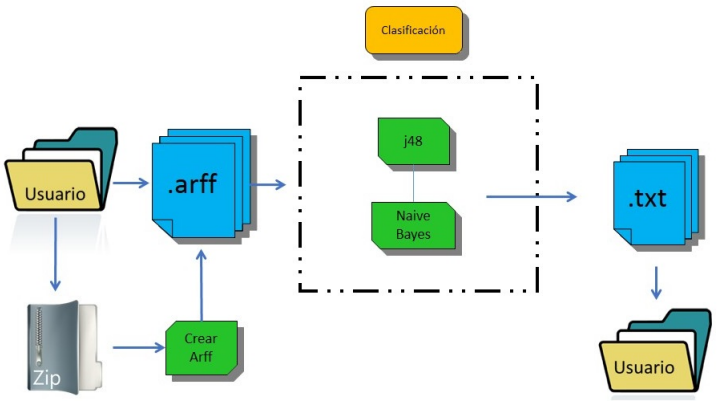


Fig. 7. Módulo de Clasificación. En la versión actual del Laboratorio en línea sólo se han incorporado dos clasificadores: Árboles de decisión y Naïve Bayes, ambos ampliamente utilizados en la comunidad científica.

De forma alternativa, en este módulo el usuario puede crear un archivo *.arff* a partir de un archivo comprimido (.zip). Lo único que necesita es que el archivo *.zip* contenga una carpeta (raíz), que a su vez contenga sub-carpetas que representen las clases de los documentos que estén contenidos en ellas. Por ejemplo, en la Figura 8, se tienen las clases $a, \ldots, n$ donde la clase a tiene los documentos $d_1, ..., d_m$ y así sucesivamente.

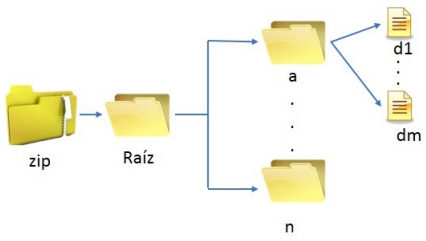


Fig. 8. Estructura del archivo zip.

Cuando se elige esta modalidad, el sistema calcula de manera interna la matriz término-documento (*i.e.,* se realiza el proceso de indexado), permitiendo llevar los documentos contenidos en las diferentes carpetas, a su representación

vectorial, siendo las instancias los documentos contenidos en las diferentes carpetas y los atributos el vocabulario de la colección. Así entonces, bajo este esquema, el *indexado* de los documentos de entrenamiento (Tr), denota la actividad de hacer el mapeo de un documento d_j en una forma compacta de su contenido. La representación más comúnmente utilizada para representar cada documento es un vector con términos ponderados como entradas, concepto tomado del modelo de espacio vectorial usado en recuperación de información [14]. Es decir, un texto d_j es representado como el vector $\overrightarrow{d}_j = \langle w_{kj}, \ldots, w_{|\tau|j} \rangle$, donde τ es el *diccionario*, *i.e.*, el conjunto de términos que ocurren al menos una vez en algún documento de Tr, mientras que w_{kj} representa la importancia del término t_k dentro del contenido del documento d_j.

En ocasiones τ es el resultado de filtrar las palabras del vocabulario, *i.e.,* resultado de un preprocesamiento. Una vez que hemos hecho los filtrados necesarios, el diccionario τ puede definirse de acuerdo a diferentes criterios, sin embargo el que se empleó en este sistema corresponde a la Bolsa de Palabras (BOW).

Cuando esta modalidad es elegida, el usuario tiene la posibilidad de elegir el esquema de pesado que quiere utilizar en la representación vectorial, es decir la importancia w_{kj} de cada término, los cuales son:

- *Ponderado Booleano:* Consiste en asignar el peso de 1 si la palabra ocurre en el documento y 0 en otro caso.

$$w_{kj} = \begin{cases} 1, \text{si } t_k \in d_j \\ 0, \text{en otro caso} \end{cases} \tag{1}$$

- *Ponderado por frecuencia de término (TF):* En este caso el valor asignado es el número de veces que el término t_k ocurre en el documento d_j.

$$w_{kj} = f_{kj} \tag{2}$$

- *Ponderado por frecuencia relativa (TF-IDF):* Este tipo de ponderado es una variación del tipo anterior y se calcula de la siguiente forma:

$$w_{kj} = TF(t_k) \times IDF(t_k) \tag{3}$$

  donde $TF(t_k) = f_{kj}$, es decir, la frecuencia del término t_k en el documento d_j. IDF es conocido como la "frecuencia inversa" del término t_k dentro del documento d_j. El valor de IDF es una manera de medir la "rareza" del término t_k. Para calcular el valor de IDF se utiliza la siguiente fórmula:

$$IDF(t_k) = \log \frac{|D|}{\{d_j \in D : t_k \in d_j\}} \tag{4}$$

  donde D es la colección de documentos que está siendo indexada.

Una vez que se ha indexado la colección, el archivo *.arff* es construido de manera automática y el proceso de clasificación se puede realizar de forma tradicional.

Es importante mencionar que cada función incluida dentro del laboratorio posee un botón de ayuda que especifica su funcionamiento y los enlaces necesarios a documentación más extensa sobre las diferentes herramientas utilizadas en este sistema. El laboratorio en línea para el procesamiento automático de documentos estará disponible en el sitio del Grupo de Lenguaje y Razonamiento de la UAM-C cuya página es: http://lyr.cua.uam.mx

4. Conclusiones y trabajo futuro

El Laboratorio en Línea para Procesamiento Automático de Documentos pretende ser una herramienta útil para realizar las diferentes tareas más comunes en el PLN. Hasta el momento, el sistema ha mostrado un funcionamiento estable con documentos de tamaño mediano. Por ejemplo, el preprocesamientos de documentos se realiza de una forma muy eficaz y rápida (segundos) en documentos de tamaño entre 10 000 y 50 000 palabras, funcionando para documentos en inglés o en español.

En el etiquetado POS, NER y *Parse Tree* por el momento solo funciona para documentos escritos en el idioma inglés . El realizar un archivo *.arff* a partir de un *.zip* es prácticamente inmediato (segundos), mientras que el proceso de de clasificación puede ser más tardado dependiendo del tamaño de la matriz proporcionada. En general, gracias al empleo del laboratorio se ha reducido el tiempo empleado para utilizar este tipo de herramientas por usuarios con poca experiencia, ya que se ha minimizado el uso de programación para realizar experimentos. También permite que usuarios sin experiencia puedan utilizar este tipo de herramientas y a través de la visualización de los resultados se ha comprobado que los usuarios que muestran mayor interés por las tareas comunes en el PLN.

Es importante mencionar que el laboratorio se desarrolló con la intención de que la comunidad de PLN en México pueda integrar más herramientas o mejorar las ya existentes. Idealmente el crecimiento del laboratorio propuesto deberá ser orientado a incluir y/o desarrollar herramientas específicas para el idioma español, lo cual permitirá impactar de manera directa en la comunidad de PLN tanto en México como a nivel internacional entre los países de habla hispana.

Agradecimientos. Agradecemos el apoyo otorgado por la Universidad Autónoma Metropolitana Unidad Cuajimalpa, al SNI-Conacyt y al Conacyt por el apoyo otorgado con el proyecto número CB2010/153315.

Referencias

1. Asociación Mexicana para el Procesamiento del Lenguaje Natural, *http://www.ampln.org* (Última visita en Julio de 2013)

2. Vallez, M., Pedraza-Jimenez, R.: El Procesamiento del Lenguaje Natural en la Recuperación de Información Textual y áreas afines [en linea]. "Hipertext.net", núm. 5, 2007. ¡http://www.hipertext.net¿
3. Manning, C.D., Schutze, H.: Foundations of statistical natural language processing. Massachusetts Institute of Technology. Second printing with corrections, 2000
4. Weka Documentation. The University of Waikato. *http://www.cs.waikato.ac.nz/ml/weka/*
5. Witten, I., Frank, E., Hall, M. A.: Data Mining: Practical Machine Learning Tools and Techniques. Morgan Kaufmann Publishers (2011)
6. Bird, S., Klein, E., Loper, E.: Natural Languaje Processing with Python. O'Reilly Media (2009)
7. Zeimpekis, D., Gallopoulos, E.: TMG: A MATLAB Toolbox for Generating Term-Document Matrices from Text Collections. (2005)
8. The Stanford Natural Language Processing Group *http://nlp.stanford.edu/index.shtml*(Última visita en Noviembre de 2013)
9. Twitter NLP and Part-of-Speech Tagging. University, Carnegie Mellon *http://www.ark.cs.cmu.edu/TweetNLP/* (Última visita en Noviembre de 2013)
10. LingPipe *http://alias-i.com/lingpipe/index.html* (Última visita en Noviembre de 2013)
11. Chaoticity *http://chaoticity.com/dependensee-a-dependency-parse-visualisation-tool/* (Última visita en Noviembre de 2013)
12. Téllez Valero, A., Montes y Gómez, M., Villaseñor Pineda, L.: Using Machine Learning for Extracting Information from Natural Disaster News Reports. Computación y Sistemas, 13(1) (2009)
13. Sebastiani, F. Machine Learning in Automated Text Categorization. En *ACM Computing Surveys*, Vol. 34, No. 1, March 2002, pp. 1–47 (2002)
14. Baeza-Yates, R., Ribeiro-Neto, B.: Modern Information Retrival, Addison Wesley (1999)

Análisis de sentimientos en Twitter: impacto de las características morfológicas

Miguel Jasso-Hernández[1,2], David Pinto[2], Darnes Vilariño[2], Cupertino Lucero[1]

[1] Universidad Tecnológica de Izúcar de Matamoros, México

[2] Facultad de Ciencias de la Computación
Benemérita Universidad Autónoma de Puebla, México

migueljhdz18@yahoo.com.mx, {dpinto, darnes}@cs.buap.mx,
cuper_lucero@hotmail.com

Resumen. En este artículo se presentan una serie de experimentos encaminados al análisis de sentimientos sobre textos escritos en Twitter. En particular, se estudian diversas características morfológicas para la representación de los textos con la finalidad de determinar aquellas que proporcionan el mejor rendimiento en el momento de detectar la carga emocional contenida en los Tweets.

Palabras clave: Análisis de sentimientos, etiquetas morfológicas, Tweets.

1. Introducción

Analizar la carga emocional en textos es una tarea que reviste una gran importancia en la actualidad. Existe una multitud de aplicaciones que pueden resultar beneficiadas de procedimientos computacionales que permitan detectar, automáticamente, si la intención del autor ha sido expresarse de manera "positiva", "negativa", "objetiva" o "neutral". Consideremos, por ejemplo, el caso de una personalidad política que requiere saber si la comunidad tiene una apreciación positiva o negativa sobre su persona. Otro ejemplo, podría ser la determinación de la reputación para una institución pública o privada. En cualquiera de los dos ejemplos, existe la necesidad de analizar el punto de vista de las personas (usuarios) sobre nuestras entidades destino. Si bien, en el pasado era común aplicar cuestionarios hacia los usuarios, en la actualidad esta práctica es raramente usada debido principalmente a los siguientes inconvenientes:

1. La aplicación de cuestionarios es un proceso costoso, desde el punto de vista del tiempo y económico.
2. El concentrado de dichos cuestionarios requiere tiempo y un análisis posterior.
3. La selección de candidatos sobre los cuales se aplica el cuestionario debe ser decidido con cuidado para garantizar que los resultados de análisis sean apropiados (cantidad y calidad).
4. El análisis de los datos tiene que hacerse con prontitud para evitar que las conclusiones sean obsoletas.

De esta manera, es mucho más práctico y conveniente usar datos frescos obtenidos directamente de las redes sociales. Las personas suelen expresarse libremente sobre los temas que son de su interés. El único problema es que dichos datos son expresados en lenguaje natural, y por tanto requieren de métodos computacionales automáticos para su tratamiento. Aun así, esta aproximación resulta ser mucho más atractiva para las empresas y ha resultado en una área de investigación sumamente activa dentro de la comunidad relacionada con el procesamiento automático del lenguaje natural.

El objetivo de este trabajo es el de evaluar el impacto en el uso de diversas características morfológicas sobre la tarea de la detección de carga emocional en Tweets (*positive*, *negative*, *neutral* y *objective*). El problema se ha planteado solucionar desde la perspectiva de la clasificación de textos usando métodos de aprendizaje automático. Esta perspectiva se ha tomado en cuenta basado en la existencia de un corpus supervisado (con clasificación manual).

A continuación se presentan algunos trabajos relacionados con el análisis automático de sentimientos en textos obtenidos desde las redes sociales.

1.1. Trabajos relacionados

Existen estudios relacionados con la identificación de emociones en Twitter, sin embargo, pocos de ellos prestan atención al análisis de la aportación parcial que generan las características morfológicas. Por ejemplo, en [1] se calcula la probabilidad de polaridad a priori asociada con etiquetas de partes de la oración (PoS). Se usan hasta 100 características adicionales que incluyen el uso de emoticones y diccionarios de palabras positivas y negativas. Los resultados reportados muestran hasta un 60 % de exactitud. Por otro lado, en [13] se propone una estrategia que hace uso de pocos recursos léxicos; en particular, utiliza relaciones discursivas tales como conectividad y condicionals para incorporarlas en los modelos clásicos de bolsas de palabras con la intención de mejorar los valores de exactitud sobre el proceso de clasificación. También se prueba la influencia de operadores semánticos tales como los modales y las negaciones y el grado en que afectan el sentimiento presente en una oración.

Uno de los mayores avances obtenidos en la tarea de análisis de sentimientos ha sido en un competencia propuesta en el marco de SemEval 2013 [3,4,5,6,7,8,9, 10,11,12,13,14,15]. Algunos de los trabajos han permitido tener un panorama amplio de diversos métodos y características usadas en la tarea mencionada. No cabe duda que esta es una tarea importante que será de atención por la comunidad de lingüística computacional en los años siguientes.

2. Experimentos

En esta sección se describen los experimentos llevados a cabo. A continuación se describe el conjunto de datos usado; posteriormente, las características evaluadas para representar a los textos; enseguida el tipo de clasificador usado; para finalmente mostrar los resultados obtenidos.

2.1. Conjunto de datos usado

En nuestros experimentos hemos hecho uso de un conjunto de datos de entrenamiento y prueba proporcionados en el marco de la competencia SemEval 2014; en particular, en la Tarea 9 que ha sido denominada "Análisis de Sentimientos en Twitter"[1]. Este corpus presenta 6,162 tweets escritos en el idioma inglés, los cuales han sido etiquetados manualmente con las siguientes clases: *positive*, *negative*, *neutral* y *objective*, los cuales pueden ser utilizados como datos de entrenamiento. El conjunto de textos de prueba contiene 978 tweets. Una descripción de sus características generales puede ser vista en la Tabla 1.

Tabla 1. Características del corpus de evaluación.

Característica	**Corpus de entrenamiento**	**Corpus de prueba**
No. de Tweets	6,162	978
No. de Palabras	100,973	15,992
Vocabulario	17,000	4,611
Longitud promedio	16.38	16.35
Tweets positivos	2,252	356
Tweets negativos	874	184
Tweets neutrales	1,062	330
Tweets objetivos	1,974	108

El vocabulario del corpus de prueba comparte 3,380 términos con el vocabulario del corpus de entrenamiento. Lo cual significa que el 73 % de su vocabulario está presente en el vocabulario usado en el entrenamiento. Sin embargo, dado el tamaño del vocabulario de entrenamiento (17,000) podemos ver que solamente un 15 % del vocabulario es común entre ambos corpora, lo cual muestra claramente que existe una gran cantidad de palabras que no serán útiles en la tarea de clasificación.

2.2. Descripción de las características usadas

Tal y como se ha mencionado, el objetivo de este trabajo consiste en evaluar el impacto que pueden tener las características morfológicas sobre el proceso de representación textual, cuando se lleva a cabo una tarea de identificación de carga emocional en Tweets. Así, hemos filtrado cada palabra de los Tweets del corpus de entrenamiento y prueba, dejando únicamente aquellas palabras que cumplan con una y solamente una etiqueta morfológica. El proceso de etiquetado se llevó a cabo usando el etiquetador TreeTagger[2]. Las etiquetas usadas pueden ser vistas en la Tabla 2.

Dado que los Tweets son textos sumamente cortos, en algunos casos resulta que la cantidad de palabras que cumple con alguna etiqueta PoS es cero, por

[1] `http://alt.qcri.org/semeval2014/task9/index.php?id=data-and-tools`
[2] `http://www.cis.uni-muenchen.de/~schmid/tools/TreeTagger/`

Tabla 2. Etiquetas morfológicas usadas en el proceso de clasificación.

Etiqueta PoS	Descripción
JJ	Adjetivo
NN	Sustantivo en singular
VBN	Verbo en pasado participio
VB	Verbo en su forma base
RB	Adverbio
IN	Intersección
NP	Nombre propio en singular
PP	Preposición
RBR	Adverbio comparativo
RBS	Adverbio superlativo
RP	Partícula
VBG	Verbo en gerundio o pasado participio
JJR	Adjetivo comparativo
JJS	Adjetivo superlativo
MD	Modal
NPS	Nombre propio en plural
PDT	Predeterminante
VBZ	Verbo en presente, tercera persona singular
VBP	Verbo en presente, no tercera persona singular
WDT	Determinante tipo Wh
WP	Pronombre tipo Wh
WPS	Pronombre posesivo tipo Wh
WRB	Adverbio tipo Wh
NNS	Sustantivo en plural

tanto se tomó la decisión de seleccionar las primeras cinco palabras en aquellos casos en los cuales la selección de palabras fuese nula. Esto permite homogeneizar los resultados haciendolos comparables a través de todas las etiquetas morfológicas, sobre todo pensando en que a corto plazo nos gustaría realizar un ensamble de los resultados por categoría morfológica. En este caso, se necesitaría que cada instancia tenga, al menos, una característica.

2.3. Clasificador usado

Se seleccionaron tres clasificadores de distinta naturaleza con la finalidad de evaluar el proceso adecuadamente. En particular, se usó un clasificador basado en árboles de decisión conocido como J48, el cual es una implementación del algoritmo C4.5, uno de los algoritmos de minería de datos más utilizado. También se uso una implementación del algoritmo de Naïve Bayes, el cual calcula la probabilidad de cada característica, dada la característica anterior. Finalmente, se empleó una implementación de las máquinas de soporte vectorial (SVM) conocida como SMO. Una SVM construye un hiperplano o conjunto de hiperplanos en un espacio de dimensionalidad muy alta que puede ser utilizado en problemas

de clasificación o regresión. El objetivo es buscar una buena separación entre las clases, lo cual permitirá un clasificación correcta.

Se usaron las implementaciones de J48, Naive Bayes y SMO presentes en la herramienta WEKA[3], empleando los parámetros default existentes en cada clasificador.

2.4. Resultados

En esta sección se presentan los resultados obtenidos sobre cada uno de los corpora (entrenamiento y prueba), usando las características comentadas anteriormente. Es importante mencionar que los experimentos se realizaron sobre cada una de las etiquetas PoS de forma independiente. La Tabla 3 muestra el porcentaje de datos que fueron clasificados correctamente (C) y el porcentaje de aquellos que fueron clasificados incorrectamente (I). Hemos resaltado los valores más altos, los cuales, son obtenidos principalmente por el clasificador basado en máquinas de soporte vectorial (SMO).

En la Figura 1 se observa que el comportamiento obtenido por SMO es superior, sin importar, el tipo de etiqueta morfológica utilizada. En particular, el orden de importancia en el proceso de clasificación es el siguiente: JJ, PP, RB, VB, VBP, NN, IN, JJS, RBR, RBS, WP, NPS, VBG, WRB, WPS, PDT, JJR, VBN, WDT, NNS, RP, NP, VBZ, MD. Es decir, de acuerdo a los resultados obtenidos en el corpus de entrenamiento se observa que las cinco características de mayor peso son: adjetivo, preposición, adverbio, verbo en su forma base y verbo en presente (no tercera persona singular).

Para el caso del corpus de prueba (ver Figura 2), se observa que existen únicamente dos casos en los cuales el algoritmo de clasificación SMO no obtiene los mejores resultados (preposición y adverbio). En estos dos casos, el algoritmo de Naïve Bayes obtiene mejores resultados que SMO. El orden de importancia de cada etiqueta morfológica varia bastante con respecto al obtenido sobre el corpus de entrenamiento. Este orden es el siguiente: NN, JJ, VBP, RB, NP, JJS, WRB, VBN, WP, MD, VBZ, VB, PDT, JJR, NNS, RBS, WPS, RP, RBR, VBG, WDT, IN, NPS, PP. Es decir, de acuerdo a los resultados obtenidos en el corpus de prueba se observa que las cinco características de mayor peso son: sustantivo en singular, adjetivo, verbo en presente (no tercera persona singular), adverbio y nombre propio en singular.

De esta manera, podemos observar que las siguientes características son las más importantes en ambos corpora (entrenamiento y prueba): adjetivo, verbo en presente (no tercera persona singular) y adverbio. Es curioso que la preposición se encuentre como una característica importante en el corpus de entrenamiento, mientras que en el corpus de prueba resultó ser la característica de menor importancia.

Estos resultados podrían ser mejorados al buscar un ensamble de características morfológicas que mejor representen a los Tweets. Se deberá hacer una análisis exhaustivo para evitar considerar características que dupliquen su grado

[3] http://www.cs.waikato.ac.nz/ml/weka/

Tabla 3. Resultados del proceso de clasificación usando solamente las características morfológicas.

	J48				Naïve Bayes				SMO			
Etiqueta	Entrenamiento		Prueba		Entrenamiento		Prueba		Entrenamiento		Prueba	
PoS	C	I	C	I	C	I	C	I	C	I	C	I
JJ	0.448	0.552	0.305	0.695	0.432	0.568	0.332	0.668	**0.467**	0.533	**0.340**	0.660
NN	0.430	0.570	0.308	0.692	0.421	0.579	0.323	0.677	**0.452**	0.548	**0.343**	0.657
VBN	0.434	0.566	0.314	0.686	0.412	0.588	0.313	0.687	**0.443**	0.557	**0.325**	0.675
VB	0.426	0.574	0.300	0.700	0.407	0.593	0.318	0.682	**0.455**	0.545	**0.322**	0.678
RB	0.445	0.555	0.318	0.682	0.424	0.576	**0.340**	0.660	**0.460**	0.540	0.337	0.663
IN	0.416	0.584	0.309	0.691	0.407	0.593	0.316	0.684	**0.452**	0.548	**0.318**	0.682
NP	0.423	0.577	0.316	0.684	0.413	0.587	0.305	0.695	**0.441**	0.559	**0.332**	0.668
PP	0.437	0.563	0.312	0.688	0.415	0.585	**0.316**	0.684	**0.464**	0.536	0.312	0.688
RBR	0.433	0.567	0.317	0.683	0.414	0.586	0.310	0.690	**0.448**	0.552	**0.319**	0.681
RBS	0.439	0.561	0.308	0.692	0.414	0.586	0.310	0.690	**0.446**	0.554	**0.321**	0.679
RP	0.428	0.572	0.300	0.700	0.404	0.596	0.317	0.683	**0.442**	0.558	**0.321**	0.679
VBG	0.432	0.568	0.315	0.685	0.412	0.588	0.305	0.695	**0.444**	0.556	**0.319**	0.681
JJR	0.436	0.564	0.313	0.687	0.414	0.586	0.310	0.690	**0.443**	0.557	**0.322**	0.678
JJS	0.442	0.558	0.305	0.695	0.415	0.585	0.312	0.688	**0.451**	0.549	**0.329**	0.671
MD	0.426	0.574	0.294	0.706	0.413	0.587	0.320	0.680	**0.440**	0.560	**0.324**	0.676
NPS	0.434	0.566	0.308	0.692	0.412	0.588	0.309	0.691	**0.445**	0.555	**0.316**	0.684
PDT	0.435	0.565	0.310	0.690	0.413	0.587	0.309	0.691	**0.443**	0.557	**0.322**	0.678
VBZ	0.435	0.565	0.300	0.700	0.414	0.586	0.322	0.678	**0.441**	0.559	**0.323**	0.677
VBP	0.435	0.565	0.302	0.698	0.417	0.583	0.328	0.672	**0.452**	0.548	**0.338**	0.662
WDT	0.429	0.571	0.309	0.691	0.413	0.587	0.312	0.688	**0.442**	0.558	**0.319**	0.681
WP	0.434	0.566	0.309	0.691	0.413	0.587	0.310	0.690	**0.445**	0.555	**0.324**	0.676
WPS	0.434	0.566	0.308	0.692	0.412	0.588	0.309	0.691	**0.444**	0.556	**0.321**	0.679
WRB	0.432	0.568	0.306	0.694	0.411	0.589	0.310	0.690	**0.444**	0.556	**0.325**	0.675
NNS	0.433	0.567	0.315	0.685	0.414	0.586	0.315	0.685	**0.442**	0.558	**0.322**	0.678

de representatividad y buscar solamente aquellas que en su conjunto representen mejor a todos los datos.

3. Conclusiones

En este trabajo se analizaron diversas características morfológicas con la finalidad de determinar el grado de importancia de cada una de ellas en la tarea de análisis de sentimientos en Twitter. Se usaron tres diferentes clasificadores sobre datos de entrenamiento y prueba, observando que las características estables a seleccionar en esta tarea deberían ser: adjetivo, verbo en presente (no tercera persona singular) y adverbio.

Como trabajo inmediato, nos proponemos investigar si el ensamble de diversas características podría ayudar a mejorar los resultados obtenidos. Es decir, nos gustaría investigar si ciertas características morfológicas son mutuamente excluyentes, o al menos bastante excluyentes, para poder representar los textos como una mezcla de características morfológicas.

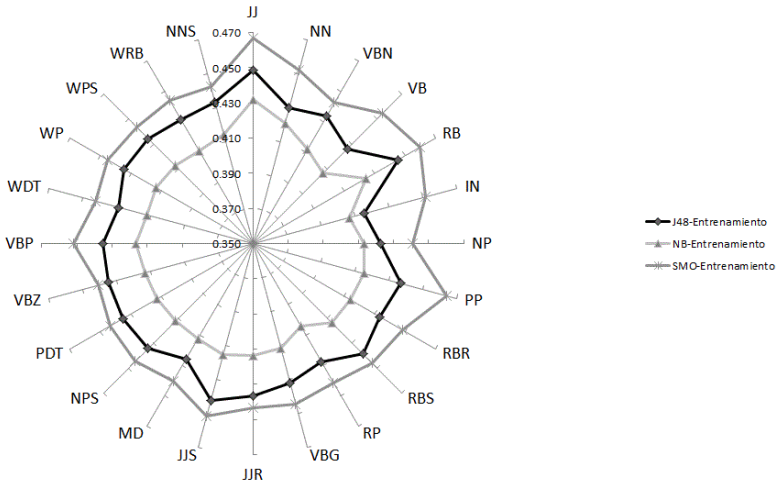


Fig. 1. Resultados obtenidos para cada etiqueta morfológica sobre el corpus de entrenamiento.

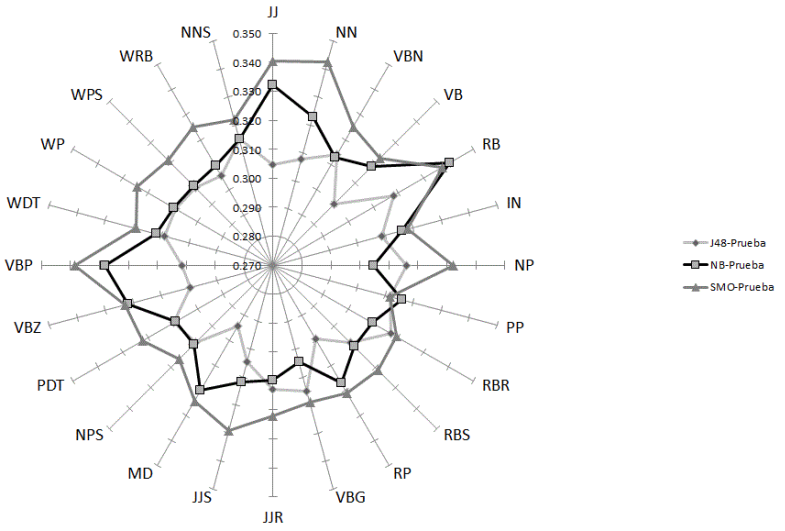


Fig. 2. Resultados obtenidos para cada etiqueta morfológica sobre el corpus de prueba.

Un aspecto interesante sería el de observar cómo se comporta un clasificador con las tres etiquetas que parecen ser las mejores para el conjunto de entrenamiento y prueba, así como las cinco etiquetas morfológicas que obtuvieron los mejores resultados para el conjunto de entrenamiento, por un lado, y por otro, las cinco que se comportaron mejor para el conjunto de prueba. En un trabajo siguiente nos planteamos hacer este análisis.

En nuestros experimentos hemos usado TreeTagger para obtener las categorías morfológicas, sin embargo, este etiquetador no está optimizado para etiquetar textos tipo Tweet, por tanto, sería conveniente hacer pruebas con un etiquetador PoS especial para Twitter como el de CMU[4].

Referencias

1. Agarwal, A., Xie, B., Vovsha, I., Rambow, O., Passonneau, R.: Sentiment analysis of twitter data. In: Proceedings of the Workshop on Language in Social Media (LSM 2011). pp. 30–38. Association for Computational Linguistics, Portland, Oregon (June 2011)
2. Balage Filho, P., Pardo, T.: Nilc_usp: A hybrid system for sentiment analysis in twitter messages. In: Second Joint Conference on Lexical and Computational Semantics (*SEM), Volume 2: Proceedings of the Seventh International Workshop on Semantic Evaluation (SemEval 2013). pp. 568–572. Association for Computational Linguistics, Atlanta, Georgia, USA (June 2013)
3. Balahur, A., Turchi, M.: Improving sentiment analysis in twitter using multilingual machine translated data. In: Proceedings of the International Conference Recent Advances in Natural Language Processing RANLP 2013. pp. 49–55. INCOMA Ltd. Shoumen, BULGARIA, Hissar, Bulgaria (September 2013)
4. Becker, L., Erhart, G., Skiba, D., Matula, V.: Avaya: Sentiment analysis on twitter with self-training and polarity lexicon expansion. In: Second Joint Conference on Lexical and Computational Semantics (*SEM), Volume 2: Proceedings of the Seventh International Workshop on Semantic Evaluation (SemEval 2013). pp. 333–340. Association for Computational Linguistics, Atlanta, Georgia, USA (June 2013)
5. Chawla, K., Ramteke, A., Bhattacharyya, P.: Iitb-sentiment-analysts: Participation in sentiment analysis in twitter semeval 2013 task. In: Second Joint Conference on Lexical and Computational Semantics (*SEM), Volume 2: Proceedings of the Seventh International Workshop on Semantic Evaluation (SemEval 2013). pp. 495–500. Association for Computational Linguistics, Atlanta, Georgia, USA (June 2013)
6. Clark, S., Wicentwoski, R.: Swatcs: Combining simple classifiers with estimated accuracy. In: Second Joint Conference on Lexical and Computational Semantics (*SEM), Volume 2: Proceedings of the Seventh International Workshop on Semantic Evaluation (SemEval 2013). pp. 425–429. Association for Computational Linguistics, Atlanta, Georgia, USA (June 2013)
7. Hamdan, H., Béchet, F., Bellot, P.: Experiments with dbpedia, wordnet and sentiwordnet as resources for sentiment analysis in micro-blogging. In: Second Joint Conference on Lexical and Computational Semantics (*SEM), Volume 2: Proceedings of the Seventh International Workshop on Semantic Evaluation (SemEval 2013). pp. 455–459. Association for Computational Linguistics, Atlanta, Georgia, USA (June 2013)
8. Han, Q., Guo, J., Schuetze, H.: Codex: Combining an svm classifier and character n-gram language models for sentiment analysis on twitter text. In: Second Joint Conference on Lexical and Computational Semantics (*SEM), Volume 2: Proceedings of the Seventh International Workshop on Semantic Evaluation (SemEval 2013). pp. 520–524. Association for Computational Linguistics, Atlanta, Georgia, USA (June 2013)

[4] `http://www.ark.cs.cmu.edu/TweetNLP/`

9. Levallois, C.: Umigon: sentiment analysis for tweets based on terms lists and heuristics. In: Second Joint Conference on Lexical and Computational Semantics (*SEM), Volume 2: Proceedings of the Seventh International Workshop on Semantic Evaluation (SemEval 2013). pp. 414–417. Association for Computational Linguistics, Atlanta, Georgia, USA (June 2013)
10. Marchand, M., Ginsca, A., Besançon, R., Mesnard, O.: [lvic-limsi]: Using syntactic features and multi-polarity words for sentiment analysis in twitter. In: Second Joint Conference on Lexical and Computational Semantics (*SEM), Volume 2: Proceedings of the Seventh International Workshop on Semantic Evaluation (SemEval 2013). pp. 418–424. Association for Computational Linguistics, Atlanta, Georgia, USA (June 2013)
11. Martínez-Cámara, E., Montejo-Ráez, A., Martín-Valdivia, M.T., Ureña López, L.A.: Sinai: Machine learning and emotion of the crowd for sentiment analysis in microblogs. In: Second Joint Conference on Lexical and Computational Semantics (*SEM), Volume 2: Proceedings of the Seventh International Workshop on Semantic Evaluation (SemEval 2013). pp. 402–407. Association for Computational Linguistics, Atlanta, Georgia, USA (June 2013)
12. Moreira, S., Filgueiras, J.a., Martins, B., Couto, F., Silva, M.J.: Reaction: A naive machine learning approach for sentiment classification. In: Second Joint Conference on Lexical and Computational Semantics (*SEM), Volume 2: Proceedings of the Seventh International Workshop on Semantic Evaluation (SemEval 2013). pp. 490–494. Association for Computational Linguistics, Atlanta, Georgia, USA (June 2013)
13. Mukherjee, S., Bhattacharyya, P.: Sentiment analysis in Twitter with lightweight discourse analysis. In: Proceedings of COLING 2012. pp. 1847–1864. The COLING 2012 Organizing Committee, Mumbai, India (December 2012)
14. Reckman, H., Baird, C., Crawford, J., Crowell, R., Micciulla, L., Sethi, S., Veress, F.: teragram: Rule-based detection of sentiment phrases using sas sentiment analysis. In: Second Joint Conference on Lexical and Computational Semantics (*SEM), Volume 2: Proceedings of the Seventh International Workshop on Semantic Evaluation (SemEval 2013). pp. 513–519. Association for Computational Linguistics, Atlanta, Georgia, USA (June 2013)
15. Tiantian, Z., Fangxi, Z., Lan, M.: Ecnucs: A surface information based system description of sentiment analysis in twitter in the semeval-2013 (task 2). In: Second Joint Conference on Lexical and Computational Semantics (*SEM), Volume 2: Proceedings of the Seventh International Workshop on Semantic Evaluation (SemEval 2013). pp. 408–413. Association for Computational Linguistics, Atlanta, Georgia, USA (June 2013)

Detección y diagnóstico de fallas en sistemas eléctricos de potencia (SEP) combinando lógica difusa, métricas y una red neuronal probabilística

César Octavio Hernández Morales, Juan Pablo Nieto González,
Elías Gabriel Carrum Siller

Corporación Mexicana de Investigación en Materiales S.A. de C.V. (COMIMSA),
Saltillo, Coahuila, México

cesarhdz@comimsa.com, juan.nieto@comimsa.com, eliascarrum@comimsa.com

Resumen. En el presente trabajo se propone un sistema para la supervisión de una red eléctrica con cambios de carga dinámicos propuesta por la IEEE. El sistema está compuesto por dos etapas. La etapa de detección utiliza un sistema de lógica difusa y la etapa de diagnóstico hace uso de las distancias Euclidianas entre líneas con el fin de generar un patrón dentro de los elementos del sistema, el cual será clasificado por una red neuronal probabilística para determinar el tipo de falla en el sistema. La combinación de estas técnicas inteligentes es para generar un sistema más robusto y seguro. Esta metodología logra la detección e identificación de fallas simétricas y asimétricas.

Palabras clave: Detección de fallas, diagnóstico de fallas, lógica difusa, métricas, red neuronal probabilística.

1. Introducción

La detección y diagnóstico de fallas tiene un papel muy importante en muchas áreas industriales en las que interviene la ingeniería. Esto debido a los complejos sistemas con los que hoy en día realizan sus procesos las industrias. Dicha complejidad se presenta por el gran número de variables que intervienen en los procesos industriales y que se deben de tomar en cuenta al momento de su monitoreo. Por lo tanto es una tarea retadora y difícil, el poder identificar las variables que se encuentran fuera de las condiciones normales de operación dentro de un proceso que está siendo monitoreado. Una alternativa para el monitoreo de un numero grande de variables es el uso de técnicas de inteligencia artificial. En el presente artículo son utilizadas para hacer una detección y diagnóstico más eficiente. El objetivo del presente trabajo es generar un sistema capaz de detectar y diagnosticar las fallas en sistemas complejos basado en los datos históricos del proceso. La propuesta se aplica al monitoreo de un sistema eléctrico de potencia (SEP) propuesto por la IEEE que considera cambios de carga dinámicos. La presente metodología es una sucesión de un trabajo realizado por [1, 2]. La metodología propuesta está compuesta por dos

pasos. El primer paso utiliza un sistema de lógica difusa para hacer la detección de las fallas y el segundo paso da el diagnóstico final al utilizar la distancia Euclidiana entre los voltajes de las líneas para cada nodo al generar un patrón para el diagnóstico. Este patrón será clasificado en los diferentes tipos de fallas por una red neuronal probabilística para obtener un sistema robusto y seguro. En la etapa de diagnóstico se realiza una comparación de la red neuronal contra la distancia Euclidiana y la de Mahalanobis para identificar la mejor técnica para realizar el diagnóstico. La organización del trabajo es de la siguiente manera. La sección 2 hace referencia a la temática del problema y se citan algunas referencias en las cuales se han utilizado las técnicas de computación suave en aplicaciones de detección y diagnóstico de fallas, en la sección 3 se presentan las herramientas matemáticas que se emplean en la presente metodología, en la sección 4 se describe el caso de estudio, la metodología se desarrolla y se presenta la evaluación de las técnicas utilizadas en la sección 5, y en la sección 6 se dan las conclusiones.

2. Estado del arte

Desde los inicios de la utilización de las máquinas, la necesidad de conocer si se encuentra trabajando adecuadamente ha sido una tarea habitual para los ingenieros que controlan los procesos, dada esta necesidad se tiene el deseo de detectar y diagnosticar las fallas. [3] presenta conceptos básicos relacionados con este campo. De acuerdo con [4], la Figura 1 muestra la metodología general empleada para la detección y diagnóstico. El autor describe los modelos de detección y diagnóstico como procesos invasivos y no invasivos. Otro punto de vista se proporciona en [5, 6] que han abordado el problema de la detección y diagnóstico clasificando los métodos en tres categorías diferentes: los modelos cuantitativos que hacen uso de modelos matemáticos, los modelos cualitativos que realizan la detección combinando la teoría de grafos con los modelos matemáticos en alguna parte del sistema, y finalmente, los modelos que hacen uso de los datos históricos para llevar a cabo un sistema de control completo.

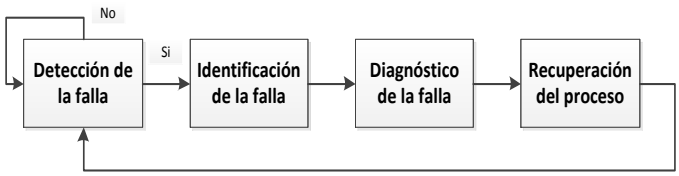


Fig. 1. Sistema de control de procesos (Adaptado de [4]).

Los métodos basados en datos históricos del proceso, son ampliamente utilizados hoy en día en la mayoría de los procesos o sistemas debido a su relativa simplicidad y eficiencia [7] y [8]. En el caso particular de los sistemas eléctricos, las técnicas de computación suave, tales como: la lógica difusa, redes neurales artificiales, razonamiento probabilístico y algoritmos evolutivos se emplean como clasificadores. [9] utiliza la lógica difusa para hacer el proceso de detección de fallas en tiempo real

en un sistema de inyección de combustible en un motor diesel. [10] propone un sistema de control de nivel de tres tanques con el uso de señales de varios sensores. El objetivo principal de este sistema es detectar señales anormales en los sensores, el uso de lógica difusa es para llevar a cabo la detección de fallas y la fase de diagnóstico se realiza al evaluar la medición de los datos proporcionados. Las distancias euclidianas se utilizan como en [11, 12]. La primera se aplica en reglas de decisión en la etapa de fotolitografía en el proceso de fabricación de un circuito integrado. El segundo calcula las distancias en un caso de estudio relacionado con los semiconductores a fin de dar el diagnóstico de las fallas. Otro tipo de métrica es la distancia de Mahalanobis, la cual es utilizada por [13] como un sistema de diagnóstico para la viscosidad de la sangre. [14] utiliza estas distancias para analizar los datos obtenidos de la vibración de los rodamientos rígidos de bolas del tipo 6205-2RS SKF, para identificar rodamientos defectuosos. [15] utiliza como herramienta de diagnóstico metodologías de inteligencia artificial tales como las redes neuronales probabilísticas, utilizadas en la detección de fallas en transformadores. Utilizando los resultados entregados por ensayos realizados sobre el aceite de un transformador, a través del análisis de los gases disueltos, [16] para el caso de un análisis de vibraciones para el diagnóstico de fallas entre máquinas de inducción utiliza una red neuronal probabilística.

3. Preliminares matemáticos

Para conocer los preliminares matemáticos correspondientes a Lógica Difusa y las Métricas ver [1, 2], puesto que ambas referencias son trabajos previos realizados por los autores.

3.1. Red neuronal probabilística

La red neuronal probabilística se compone de dos capas y parte de un vector de datos p expresado de la siguiente manera.

$$\{p_1, p_2, .., p_Q, \} \tag{1}$$

Se tiene una matriz W'de pesos transpuestos y un vector de tendencia b para una capa cuando:

$$W' = \begin{bmatrix} w_1^T \\ w_2^T \\ \vdots \\ w_s^T \end{bmatrix} = \begin{bmatrix} p_1^T \\ p_2^T \\ \vdots \\ p_Q^T \end{bmatrix}, b' = \begin{bmatrix} R \\ R \\ \vdots \\ R \end{bmatrix} \tag{2}$$

La fila de W' representa un vector prototipo que se desea reconocer y cada elemento de b' es un conjunto igual para el número de elementos en cada vector de entrada (R). Así la salida para la primera capa es:

$$a^1 = W'p + b' = \begin{bmatrix} p_1^T p + R \\ p_2^T p + R \\ \vdots \\ p_Q^T p + R \end{bmatrix} \quad (3)$$

La segunda capa es una capa competitiva. Las neuronas en esta capa, se inicializan con las salidas de la capa anterior hacia delante. Que inician el reconocimiento entre los patrones prototipo y el vector de entrada. Estas neuronas compiten con cada una de las otras para determinar un ganador. Después, una sola neurona tendrá una salida distinta a cero. La neurona ganadora indica que categoría está presentando la entrada para la red (cada vector prototipo representa una categoría).

La salida a^1 de la primera capa es usada para iniciar la segunda capa

$$a^2(0) = a^1 \quad (4)$$

La salida de esta segunda capa es actualizada de acuerdo a la siguiente relación de recurrencia.

$$a^2(t+1) = poslin(W^2 a^1(t)) \quad (5)$$

Los pesos de la segunda capa W^2 se establecen para que los elementos de la diagonal sean 1 y los elementos que se encuentran fuera de la diagonal sean valores negativos [17].

$$W_{ij}^2 = \begin{Bmatrix} 1, si\ i = j \\ -\varepsilon,\ \ en\ otro\ caso \end{Bmatrix}, cuando\ 0 < a < \frac{1}{S-1} \quad (6)$$

4. Descripción de la metodología

En este trabajo se propone una nueva metodología para realizar la detección y diagnóstico de las fallas en un SEP propuesto por la IEEE. La Figura 2 muestra la arquitectura de la propuesta. La metodología consiste en dos pasos. En el primer paso se realiza el proceso de detección, utiliza para ello un sistema de lógica difusa que evalúa las condiciones de funcionamiento del sistema. El segundo paso arroja el diagnóstico final mediante el empleo de una red neuronal probabilística. En el segundo paso se obtiene un patrón de los comportamientos del sistema tanto el modo normal así como del modo anormal de operación entre las tensiones de cada uno de los 24 nodos que forman el sistema eléctrico. Por lo tanto cuando una falla está presente la red neuronal clasifica las distancias euclidianas para dar el diagnóstico correcto e identificar qué línea o líneas presenta una falla ya sea simétrica o asimétrica.

Los pasos de la propuesta se resumen de la siguiente manera:

1. Obtener bases de datos del sistema para el modo de operación normal y para los diferentes tipos de falla que se puedan presentar en él.

2. Generar un sistema difuso que evalúa cada uno de los nodos del sistema. Se sensibiliza el sistema difuso para entregar un valor de 0.5 cuando el sistema monitoreado se encuentra en modo de falla. En caso de que el sistema se encuentre en modo de operación normal, el sistema difuso entrega un valor distinto de 0.5 y se toma otro conjunto de datos por probar.

Si el sistema difuso encuentra presencia de falla, se procede a calcular las distancias Euclidianas y con el patrón generado por estas distancias se entrena la red neuronal probabilística.

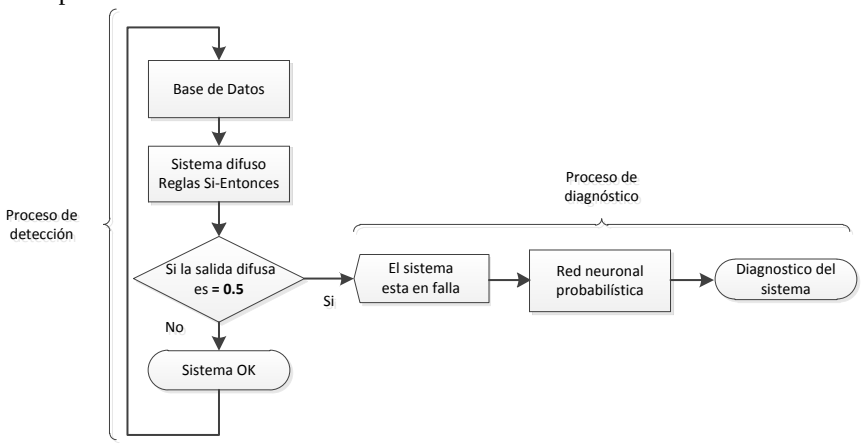


Fig. 2. Metodología general para la detección y diagnóstico de fallas.

3. La red neuronal después de ser entrenada clasifica las distancias y las asigna a un grupo y con ello da un diagnóstico mostrando qué línea se encuentra en modo de falla.

5. Caso de estudio

La presente investigación fue encaminada al monitoreo de un SEP con presencia de cambios dinámicos de carga propuesto por la IEEE. El sistema consiste de 24 nodos con sus correspondientes 3 líneas cada uno. Lo cual es un total de 72 variables por monitorear. La Figura 3 muestra dicho sistema eléctrico.

Para el proceso de detección se emplea un sistema difuso. Dicho sistema se entrenó únicamente con bases de datos en modo de operación normal. Esto representa una gran ventaja, ya que no fue necesario entrenar a 24 diferentes sistemas difusos, ni aprender los diferentes modos de falla que se pudieran presentar en el SEP. Para las simulaciones se consideraron dos tipos de fallas. Fallas simétricas y asimétricas, las primeras se producen al juntarse dos líneas entre sí. Las segundas se producen cuando una o más líneas caen a tierra. Las fallas se presentan en la figura 4.

La metodología propuesta se aplicó de la siguiente manera.

El primer paso es el proceso de detección, el cual está compuesto por los bloques mostrados en la Figura 5.

1. El primer bloque indica que el primer paso es la adquisición y análisis de los datos históricos del sistema eléctrico que se ha descrito anteriormente. El análisis llevo a cabo la evaluación de las amplitudes de las tensiones de las líneas de cada nodo. Se tomaron ventanas de 50 datos con muestras en modo de operación normal y en el modo de falla.

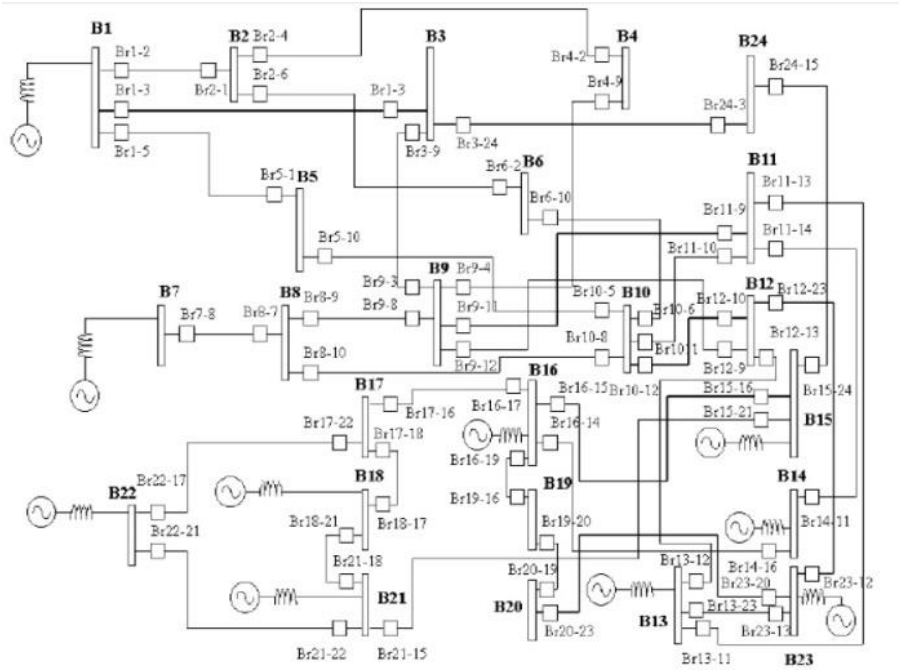


Fig. 3. Diagrama unifilar del sistema propuesto por el IEEE (Adaptado de [18]).

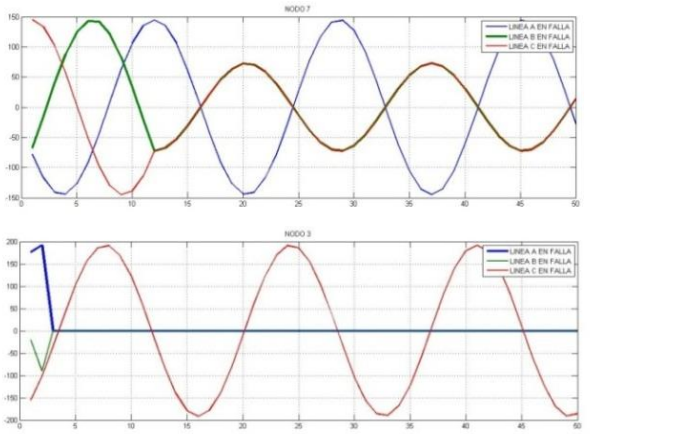


Fig. 4. Falla Simétrica y Asimétrica.

El siguiente bloque contiene las reglas difusas de los pasos 2 a 6 que se describen a continuación.

En este bloque es necesaria la selección de una función de pertenencia. Se selecciona funciones triangulares debido a su simplicidad.

2. Con las bases de datos tomados en el paso 1, se generaron las reglas difusas que describen el comportamiento del sistema en modo de operación normal. Para lo anterior se generaron un total de 76 reglas difusas que simulan el comportamiento en modo de operación normal del sistema como se muestra en la Tabla 1.
3. Los operadores difusos e implicación difusa seleccionados son los representados en las ecuaciones (2) a (6) de [1, 2]. Lo anterior debido a que dichos operadores e implicaciones pudieron explicar relativamente bien el comportamiento del sistema.

El rango considerado para las amplitudes de las tensiones observadas en cada línea es de -200 a 200 volts.

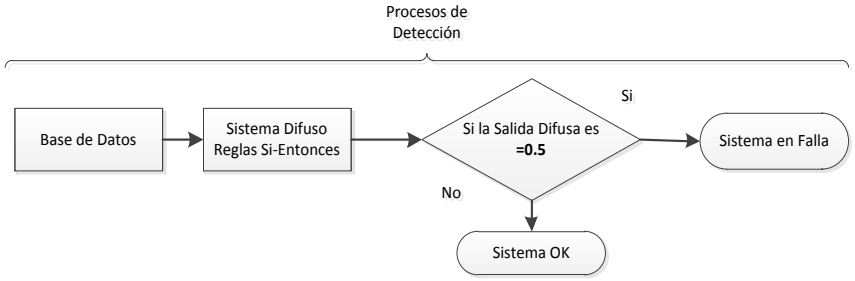


Fig. 5. Metodología del proceso de detección de fallas.

Tabla 1. Reglas difusas.

REGLA	LINEAS		
N°	L1	L2	L3
1	150,200	-150,200	0,-50
2	150,200	-150,200	0,50
3	50,100	-150,200	50,100
⋮	⋮	⋮	⋮
74	50,100	-100,150	50,100
75	0,50	-100,150	100,150
76	0,-50	-100,150	100,150

Se ha seleccionado el método centroide para llevar a cabo la etapa de defuzzificación descrita en la ecuación (7) de [1] y [2]. Este método es seleccionado debido a que es de los más comúnmente utilizados en el estado del arte.

4. La salida del sistema difuso se sensibiliza para arrojar un valor de 0.5 cuando el sistema se encuentra en un modo de falla. En caso contrario, dará una salida distinta de 0.5 indicando así un modo de operación normal del SEP.

5. Si se determina que existe una falla se procederá a realizar el segundo paso, si no, se considera que el sistema esté en un modo de operación normal y se procede a analizar la siguiente ventana de datos.

El segundo paso es el proceso de diagnóstico, el cual se lleva a cabo como se describe en las Figuras 6, 7 y 8.

Después de analizar la condición de operación del sistema y una vez detectada la presencia de una falla se procede a encontrar cual es el tipo de falla y en qué elementos se encuentra presente. Tal búsqueda se fundamentó en una primera instancia en la relación presente en las distancias entre las amplitudes de los elementos que se encuentran en modo de falla. A continuación se describen estas relaciones, primero se especifica cómo se lleva a cabo para la presencia de una falla simétrica y posterior mente para una falla asimétrica.

- Fallas simétricas

Para el caso de la falla simétrica la falla no se puede presentar en una sola línea, ya que la falla se presenta cuando se unen dos líneas entre sí. Como un ejemplo se presenta

– Dos líneas en falla simétrica.

Si las distancias entre las líneas L1-L2 es diferente a L2-L3 y L1-L3 y L2-L3 y L1-L3 son iguales hay una falla a L1 y L2, y este patrón se repite para los demás casos, como se muestra en la Figura 6.

Por lo tanto, si se cumple alguna de estas condiciones propuestas en esta sección del segundo paso, con las distancias obtenidas por medio del cálculo de las distancias Euclidianas y de Mahalanobis, se determina qué se tiene una falla simétrica presente en él sistema.

- Fallas asimétricas o a tierra

– Una de las líneas con falla a tierra.

Si al comparar las distancias entre L1-L2-L3, las distancias de L2-L3 son valores iguales, entonces la falla está presente en la línea L1. Esto se realiza de manera similar para las demás líneas.

– Dos líneas en falla a tierra.

Si las distancias entre las distancias L1-L2 son 0 y las distancias L2-L3 y L1-L3 son iguales hay una falla a tierra en L1 y L2, como se muestra en la Figura 7.

Por lo tanto, si se cumple alguna de estas condiciones propuestas en esta sección del segundo paso, para las distancias obtenidas por medio del cálculo de las distancias Euclidianas y de Mahalanobis, se determina qué se tiene una falla asimétrica presente en él sistema.

Una de las problemáticas que se tiene al utilizar estas métricas es que con ellas para realizar un correcto diagnostico se tienen que tener todos los elementos de la

muestra con presencia de falla ya que si estos no presentan este comportamiento, el patrón generado no se cumplirá. Para hacer un diagnóstico más rápido y no depender de una ventana de datos con presencia de falla en todos sus elementos, se utilizó el patrón generado por la distancia Euclidiana entre líneas para caracterizar las fallas en el sistema, y con ello se propuso entrenar una red neuronal probabilística para hacer la clasificación de la falla presente e identificar cual línea tiene la presencia de falla, con el fin de hacer el proceso de detección y diagnóstico más eficiente y evitar la presencia de falsas alarmas. La metodología propuesta para la clasificación utiliza una red neuronal la como se muestra en la Figura 8.

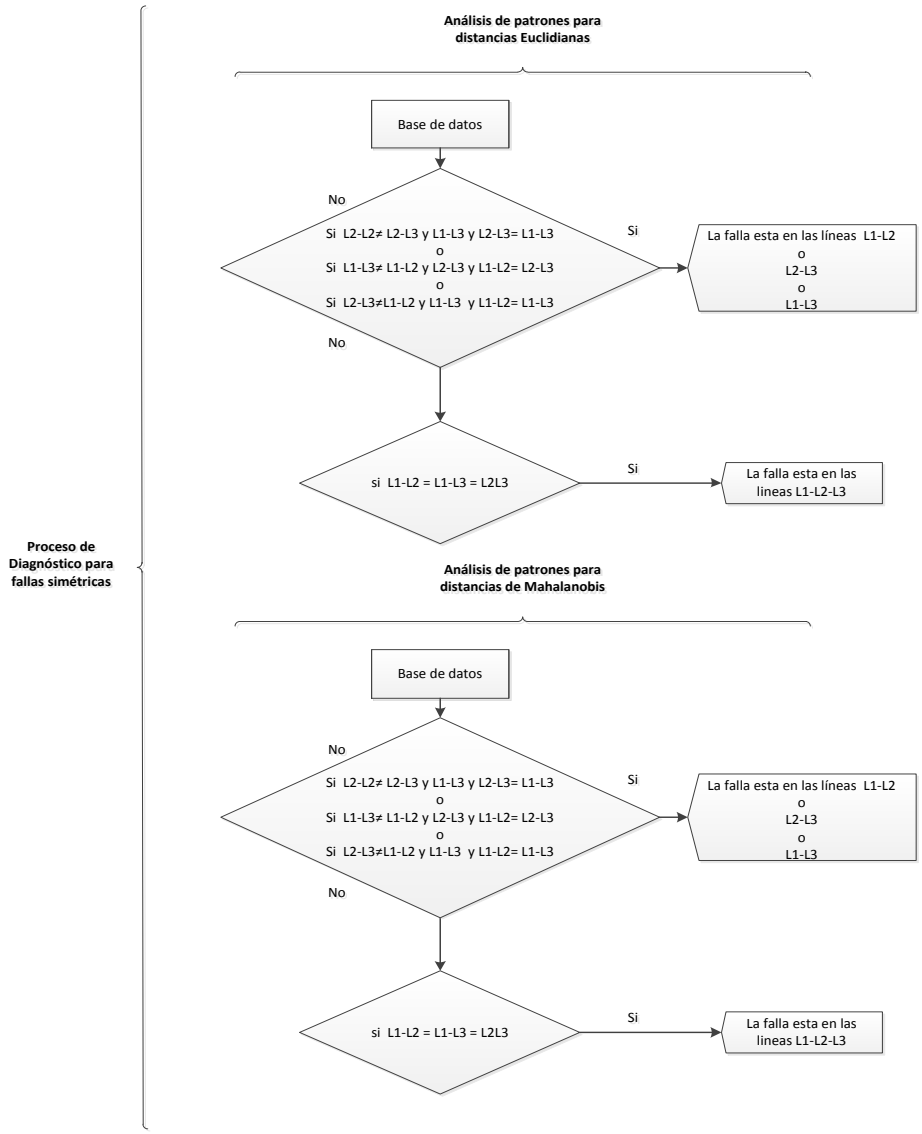


Fig. 6. Metodología para el diagnóstico de fallas Simétricas.

En la Tabla 2 se valida la metodología descrita. El sistema logra detectar la presencia de fallas en un 100% de las veces, pero en la etapa de diagnóstico el porcentaje decrece en las distancias debido a que la metodología necesita tener presente en la ventana de 50 datos, todos los datos en modo de falla para hacer un correcto diagnóstico.

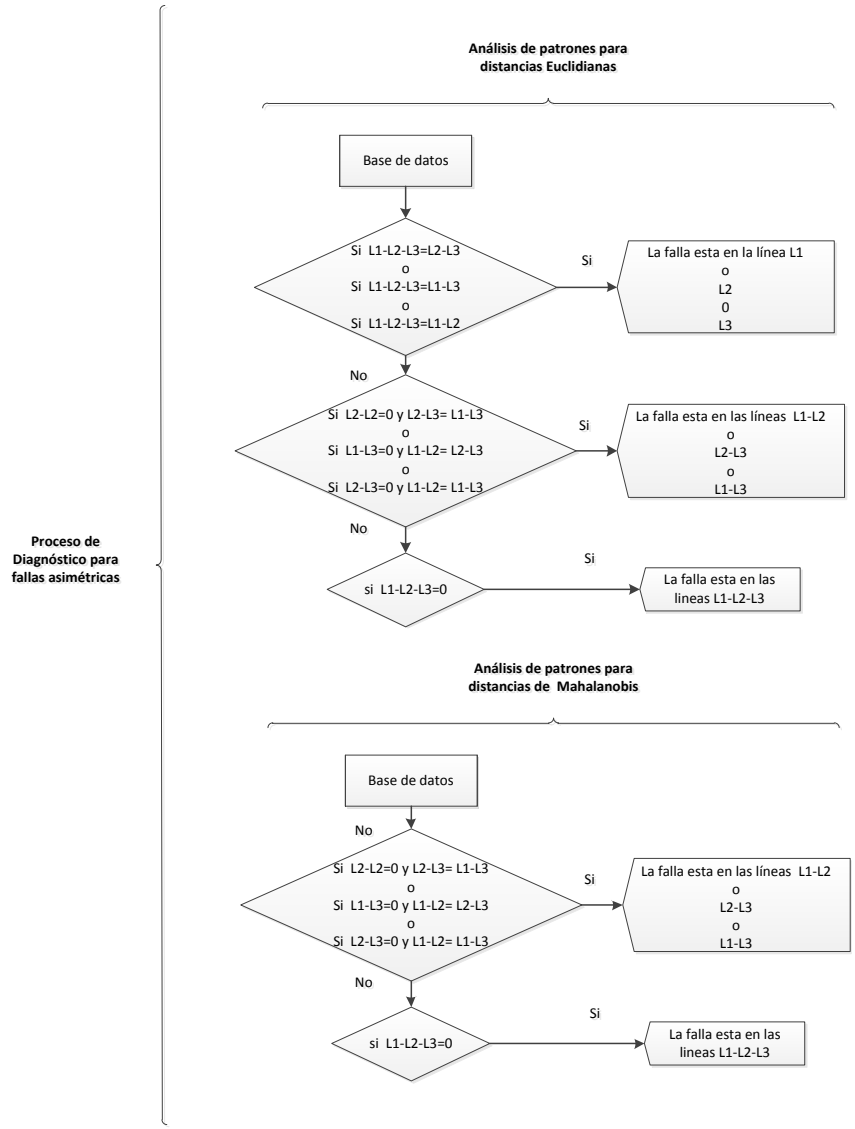


Fig. 7. Metodología para el diagnóstico de fallas Simétricas y Asimétricas.

Como se aprecia en la tabla para evitar esta deficiencia en las métricas, se genera un diagnóstico más preciso del 99% utilizando una red neuronal probabilística para la clasificación, este porcentaje es alcanzado al utilizar la distancia euclidiana entre

líneas y con ello se genera un patrón que caracteriza de una mejor manera las fallas y en qué línea están presentes dichas fallas. La red neuronal se entrenó con las distancias con ventanas de 50 datos, como se realizó con el sistema difuso y de igual forma la red neuronal se replicó en los demás nodos del sistema eléctrico y con ello se facilita la detección de fallas múltiples del sistema, la red se validó con 120 datos los cuales presentan los dos tipos de falla en sus líneas así como el modo de operación normal. La red neuronal que se generó cuenta con cuatro entradas para cada una de las distancias euclidianas obtenidas, una capa oculta la cual es entrenada con las distancias que presentan un comportamiento en modo normal de operación y de los casos de falla que se han presentado en el sistema. Una segunda capa con 12 neuronas la cual realiza la clasificación con respecto a la falla presente y una salida que da a conocer el diagnostico requerido, la red neuronal se presenta en la Figura 9.

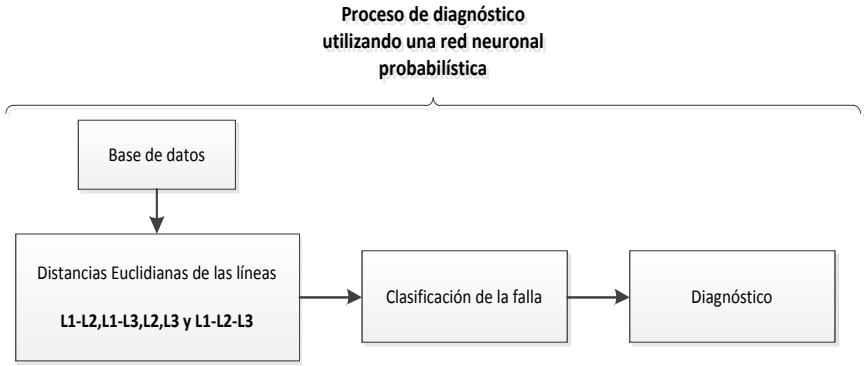


Fig. 8. Metodología para el diagnóstico utilizando una red neuronal probabilística.

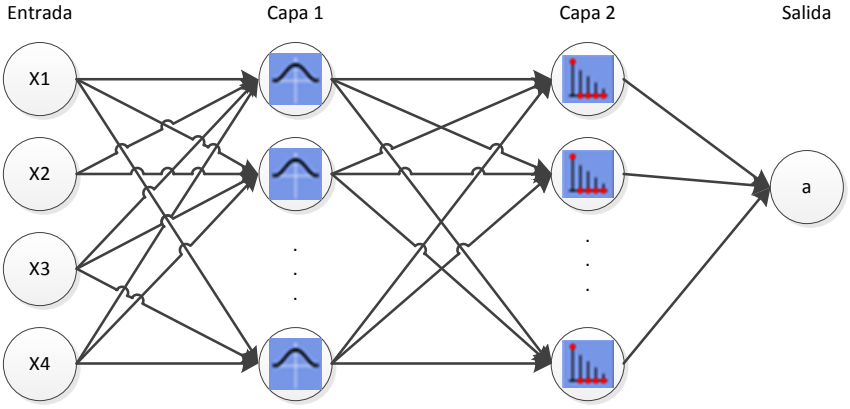


Fig. 9. Red Neuronal Probabilística.

Al determinar el tipo y elementos donde se encuentra la falla en un paso posterior se procederá a hacer la recuperación del sistema para llevarlo nuevamente a su modo normal de operación.

Tabla 2. Pruebas para validación de la metodología propuesta

N° Fallas en Ventanas de 50 Datos	% en que se realizó la Detección	% para el Diagnostico Distancia Euclidiana	% para el Diagnostico Distancia de Mahalanobis	% para el Diagnostico Con la Red Neuronal Probabilística
1	100	75	70	99
2	100	80	75	99
3	100	80	80	99
4	100	80	80	99
5	100	80	80	99

6. Conclusiones

En el presente artículo se ha propuesto un sistema de detección y diagnóstico de fallas basado en datos del histórico del proceso. La propuesta utiliza un sistema de lógica difusa para el proceso de detección y al obtener las distancias Euclidianas genera un patrón para el entrenamiento de una red neuronal probabilística.

Con la red se clasifican los patrones generados por estas distancias para arrojar un diagnóstico final más confiable y con ello disminuir la presencia de falsas alarmas. La metodología propuesta fue validada en un sistema eléctrico de potencia con cambios dinámicos de carga. El SEP monitoreado está compuesto por 24 nodos y es propuesto por la IEEE.

Como se mostró en la tabla 2 se obtiene un diagnóstico más eficiente al utilizar una herramienta inteligente para la clasificación que al utilizar una comparación entre dos tipos de métricas.

Referencias

1. César Octavio Hernández Morales and Juan Pablo Nieto González: Fault detection and diagnosis of electrical networks using a fuzzy system and euclidian distance. In: 12th Mexican International Conference on Artificial Intelligence, MICAI 2013 Mexico City, Mexico, Proceedings, Part II, pp. 216–224 (2013)
2. César Octavio Hernández Morales and Juan Pablo Nieto González: Detección y Diagnóstico de Fallas en SEP's Combinando Lógica difusa con distancias Euclidianas y de Mahalanobis. En: X Congreso Internacional sobre Innovación y Desarrollo Tecnológico, Cuernavaca, Morelos, México (2014)
3. Julio César Ramírez Valenzuela: Diagnóstico de fallos en sistemas industriales basado en razonamiento borroso y posibilístico. Universidad Politécnica de Valencia, Departamento de Ingeniería de Sistemas y Automática (2007)
4. Ali Ajami, Mahdi Daneshvar: Data driven approach for fault detection and diagnosis of turbine in thermal power plant using Independent Component Analysis (ICA). Electrical Power and Energy Systems, pp. 728–735 (2012)

5. Venkatasubramanian V., Rengaswamy R., Yin K., Kavuri S.: A review of process fault detection and diagnosis Part I. Computers and Chemical Engineering, 27, pp. 293–311 (2003)
6. Juan Pablo Nieto González, Luis Garza Castañón y Rubén Morales Menéndez: Multiple Fault Diagnosis in Electrical Power Systems with Dynamic Load Changes Using Probabilistic Neural Networks. Computación y Sistemas, 14(1), pp. 17–30 (2010)
7. Cox, Earl: Fuzzy fundamentals. IEEE Spectrum, pp. 58–61 (2002)
8. J.P. Nieto: Multiple Fault Diagnosis in Electrical Power Systems with Dynamic Load Changes Using Soft Computing. In: 11th Mexican International Conference on Artificial Intelligence, MICAI 2012: Advances in Computational Intelligence Proceedings Part 2, pp. 319–330 (2012)
9. Yong He and Lei Feng: Diesel Fuel Injection System Faults Diagnosis Based on Fuzzy Injection Pressure Pattern Recognition. In: Proceedings of the 5th World Congress on Intelligent Control and Automation, Hangzhou, China, pp. 1654–1657 (2004)
10. Seda Postalcio Lu: Signal Processing and Fuzzy Cluster Based Online Fault Diagnosis. pp. 1454–1459 (2009)
11. Ghislain Verdier and Ariane Ferreira: Adaptive Mahalanobis Distance and k-Nearest Neighbor Rule for Fault Detection in Semiconductor Manufacturing. IEEE Transactions on Semiconductor Manufacturing, 24(1), pp. 59–67 (2011)
12. Han Ya-juan, HE Zhen and SONG Guo-fang: Research for Multidimensional Systems Diagnostic Analysis based on Improved Mahalanobis Distance. 2009, IEEE, pp. 213–217.
13. Jinshan Lin and Qian Chen: Fault diagnosis of rolling bearings based on multifractal detrended fluctuation analysis and Mahalanobis distance criterion. Mechanical Systems and Signal Processing, pp. 515–533 (2013)
14. Ming Luo, Ying Zheng and Shujie Liu: Data-based Fault-tolerant Control of the Semiconductor Manufacturing Process based on K-Nearest Neighbor Nonparametric Regression. In: Proceedings of the 10th World Congress on Intelligent Control and Automation, pp. 3008–3012 (2012)
15. Juan Andrés Cadena, Juan Mauricio Cadena y Sandra Milena Pérez: Aplicación de redes neuronales probabilísticas en la detección de fallas incipientes en transformadores. Scientia et technica, vol. XIV, núm. 39, pp. 48–53 (2008)
16. Jeevanand Seshadrinath, Bhim Singh, B. K. Panigrahi: Vibration analysis based interturn fault diagnosis in induction machines. IEEE TRANSACTIONS ON INDUSTRIAL INFORMATICS, VOL. 10, NO. 1, pp. 340–350 (2014)
17. Martin T. Hagan: Neural Network design. Oklahoma State University, Mark Beale MHB, Inc., pp. 1–15 (1996)
18. Juan Pablo Nieto González: Complex Systems Fault Diagnosis Using Soft Computing and Statistical Methods. Instituto Tecnológico y de Estudios Superiores de Monterrey, Campus Monterrey, Escuela de Ingeniería y Tecnologías de Información (2012)

Memorias asociativas en la agricultura: un caso práctico

Mario Aldape Pérez, José Antonio Estrada Pavía, Joel Omar Juarez Gambino

Centro de Innovación y Desarrollo Tecnológico en Cómputo (CIDETEC),
Instituto Politécnico Nacional (IPN),
Ciudad de México, México

mario@aldape.org.mx; jaestradap@gmail.com; omarjg82@gmail.com

Resumen. Las memorias asociativas tienen una serie de características, incluyendo un rápido y eficiente método de clasificación, así como una tolerancia intrínseca al ruido que las hace ideales para gran variedad de aplicaciones. En este artículo se utilizarán las memorias alfa-beta autoasociativas con el propósito de recomendar un herbicida en base a diferentes parámetros. El objetivo de este trabajo es presentar las memorias alfa-beta como una posibilidad para resolver problemas reales en la agronomía.

Palabras clave: Memorias asociativas Alfa-Beta, herbicidas, reconocimeinto de patrones.

1. Introducción

Los sistemas agrícolas relacionados con herbicidas son complejos, en algunas ocasiones pueden ser considerados como sistemas mal definidos ya que es difícil, en este tipo de sistemas, cuantificar las relaciones entre la entrada y la salida, esta basta cantidad de relaciones existentes se debe a que la mayor parte de sistemas agrícolas involucran gran cantidad de variables diferentes. Desde 1997 los sistemas de control inteligente han sido de las tecnologías mas prosperas en el campo de los sistemas complejos [1]. A continuación se muestra una pequeña reseña de la evolución de métodos de Inteligencia Artificial (IA) usados en aplicaciones agronómicas.

En 1997 las redes neuronales artificiales (ANN) demostraron ser una alternativa eficaz para su utilización en los sistemas complejos, sus principales características son su alta capacidad de aprendizaje y su capacidad de identificar y modelar una compleja relación no lineal entre la entrada y la salida del sistema [1].

Por otro lado en 2001, un nuevo método de clasificación de redes neuronales fue presentado en este campo Self-Organizing Map (SOM), este método logra una convergencia rápida y buena generalización. La clasificación del método propuesto se probó superior, en aplicaciones agrícolas, comparado con otros clasificadores estadísticos y neurológicos [2].

La ciencia sigue cambiando y avanzando día con día, también nuevos métodos se desarrollan en el campo de la inteligencia artificial aplicada en la agricultura,

uno de estos métodos fue una red neuronal auto organizada, este método presentado en 2003 probó que las redes neuronales pueden discriminar entre diferentes especies de plantas con una precisión superior al 75 %, sin que se halla definido anteriormente las características de la planta.[3].

No sólo las ANN se han utilizado en los sistemas agrícolas, también las Máquinas de Soporte Vectorial (SVM) han demostrado ser un método útil en la IA, en 2006 las SVM demostraron que pueden ser usadas como una herramienta para la clasificación de imágenes hiperespectrales. Este método logró más de un 93 % de eficiencia clasificando datos de prueba que no habían sido enseñados [4].

Más recientemente, en 2011 el Generalized Softmax Perceptron (GSP) una arquitectura de red neuronal, se usó en conjunto con el Posterior Probability Model Selection (PPMS) en una selección compleja para clasificar las imágenes en girasoles o no girasoles [5].

Algunos autores describen las memorias asociativas como redes neuronales sin pesos, pero las memorias asociativas tienen otras cualidades que las hacen perfectas para una gran cantidad de problemas. El propósito fundamental de una memoria asociativa es recuperar patrones completos a partir de patrones de entrada, que pueden contener ruido aditivo, sustractivo o mixto [6]. Las memorias asociativas tiene dos fases funcionales, la fase de aprendizaje y la fase de recuperación, en la fase de aprendizaje la memoria asociativa es construida con un conjunto de patrones clasificados un patrón de entrada y otro un patrón de salida.

Una de las referencias más antigua de memorias asociativas data de 1961 cuando Karl Steinbuch desarrolló el modelo de memoria asociativa Lernmatrix [7] una memoria heteroasociativa capaz de trabajar como un clasificador de patrones binarios. Luego, en 1972, el modelo Linear Associator se presentó de forma independiente por Anderson [8] y Kohonen [9]. Pero el trabajo más relevante de la memoria asociativa fue desarrollado por Hopfield en 1982 [10] su modelo demuestra la interacción de elementos simples de procesamiento, similares a las neuronas, dando lugar a la aparición de propiedades computacionales colectivas, como la estabilidad de las memorias [6] sin embargo, este modelo tuvo inconvenientes uno de ellos es la limitada capacidad de recuperación siendo esta de sólo 0.015 n, donde n es la dimensión de los patrones almacenados [6].

El modelo de memoria asociativa usado en este documento se llama memoria alfa-beta, la mayor ventaja de esta memoria es la recuperación completa del conjunto fundamental.

2. Memorias asociativas Alfa-Beta

El propósito de la memoria asociativa es la correcta recuperación de patrones de salida relacionados con patrones de entrada, que pueden ser alterados con ruido aditivo, sustractivo o mezclado. Los conceptos que se utilizan en esta sección se presentan en [6].

Tabla 1. Operadores Alfa Beta.

$\alpha : A \times A \longrightarrow B$

x	y	α(x,y)
0	0	1
0	1	0
1	0	2
1	1	1

$\beta : B \times A \longrightarrow A$

x	y	β(x,y)
0	0	0
0	1	0
1	0	0
1	1	1
2	0	1
2	1	1

Una memoria asociativa M es un sistema que relaciona un patrones de entrada y salida de la siguiente forma: $x \longrightarrow \boxed{\mathbf{M}} \longrightarrow y$ con x e y, respectivamente, como los vectores-patrones de entrada y salida. Cada vector entrada esta relacionado con su correspondiente vector salida. Para cada k entero positivo, la asociación será denotada como: (x^k, y^k). La memoria asociativa M es representada por una matriz de la cual su ij-esima componente es m_{ij} [9]. La memoria M es generada *anticipadamente* de un conjunto finito de asociaciones conocidas, llamado conjunto fundamental de asociaciones. Si μ es un índice, el conjunto fundamentel es representado como: $\{(x^\mu, y^\mu) \mid \mu = 1, 2, ..., p\}$ con p como la cardinalidad del conjunto. Los patrones del conjunto fundamental se llaman patrones fundamentales. Si definimos $x^\mu = y^\mu \ \forall \mu \in \{1, 2, ..., p\}$ M es autoasociativa, de otra forma esta es heteroasociativa; en este caso, es posible establecer que $\exists \mu \in \{1, 2, ..., p\}$ para cada $x^\mu \neq y^\mu$. Si consideramos el conjunto fundamental de patrones $\{(x^\mu, y^\mu) \mid \mu = 1, 2, ..., p\}$ donde n y m son la dimensión de los patrones de entrada y salida, respectivamente, decimos que $x^\mu \in A^n$, $A = \{0, 1\}$ y $y^\mu \in A^m$. Entonces la j-esima componente del patrón de entrada es $x^\mu_j \in A$. De la misma forma, la j-esima componente de una patrón de salida es representada como $y^\mu_j \in A$. Una versión distorsionada de un patrón x^k para ser recuperado será denotado como $\widetilde{x}^k$. Si se presenta un patrón de entrada desconocido x^ω con $\omega \in \{1, 2, ..., k, ..., p\}$ a una memoria asociativa M, de tal forma que la salida corresponde exactamente con el patrón asociado y^ω, se dice que la recuperación es perfecta.

La memoria asociativa Alfa-Beta matemáticamente esta basada en dos operadores binarios: α y β. El operador Alfa es usado en la fase de aprendizaje mientas que el operador Beta es usado en la fase de recuperación. Las propiedades matemáticas de estos operadores, permiten que las memorias asociativas $\alpha\beta$ poseer características similares a la versión binaria de las memorias asociativas morfológicas, en el sentido de: la capacidad de aprendizaje, la memoria es robusta para soportar cierto tipo y cantidad de ruido y posee las características necesarias para una recuperación perfecta [11]. Primero, definimos un conjunto $A = \{0, 1\}$ y un conjunto $B = \{00, 01, 10\}$, entonces los operadores α y β pueden ser definidos en la Tabla 1.

Estos dos operadores binarios junto con el operador máximo ($\vee$) y mínimo ($\wedge$) establecen las herramientas matemáticas que le conciernen al modelo Alfa-Beta.

Las definiciones de α y β presentadas en la Tabla 1, implican que: α incrementa a la izquierda y decrementa a la derecha, β incrementa a la izquierda y a la derecha, β es el inverso izquierdo de α. De acuerdo al tipo de operador que es usado en la fase de aprendizaje, dos tipos de memorias asociativas Alfa-Beta pueden ser obtenidas. Si el operador máximo ($\vee$) es usado, una Memoria Asociativa Alfa-Beta de tipo MAX será obtenida, llamemosla M; de manera análoga, si el operador mínimo ($\wedge$) es usado, la Memoria Asociativa Alfa-Beta tipo min será obtenida, será llamada W [6]. En cualquier caso, los patrones fundamentales de entrada y salida son representados de la siguiente forma:

$$x^\mu = \begin{pmatrix} x_1^\mu \\ x_2^\mu \\ \vdots \\ x_n^\mu \end{pmatrix} \in A^n \qquad y^\mu = \begin{pmatrix} y_1^\mu \\ y_2^\mu \\ \vdots \\ y_m^\mu \end{pmatrix} \in A^m$$

Para entender como se llevan a cabo las fases de aprendizaje y recuperación, serán definidas algunas operaciones matriciales.

α máx Operación: $P_{mxr} \nabla_\alpha Q_{rxn} = \left[f_{ij}^\alpha\right]_{mxn}$, donde $f_{ij}^\alpha = \vee_{k=1}^r \ \alpha(p_{ik}, q_{kj})$
β máx Operación: $P_{mxr} \nabla_\beta Q_{rxn} = \left[f_{ij}^\beta\right]_{mxn}$, donde $f_{ij}^\beta = \vee_{k=1}^r \ \beta(p_{ik}, q_{kj})$
α mín Operación: $P_{mxr} \Delta_\alpha Q_{rxn} = \left[f_{ij}^\alpha\right]_{mxn}$, donde $f_{ij}^\alpha = \wedge_{k=1}^r \ \alpha(p_{ik}, q_{kj})$
β mín Operación: $P_{mxr} \Delta_\beta Q_{rxn} = \left[f_{ij}^\beta\right]_{mxn}$, donde $f_{ij}^\beta = \wedge_{k=1}^r \ \beta(p_{ik}, q_{kj})$

Siempre que un vector columna de dimensión m es operado con un vector fila de dimensión n, ambas operaciones ∇_α y Δ_α, son representadas por $\oplus$; consecuentemente, la siguiente expresión es valida:

$$y \nabla_\alpha x^t = y \oplus x^t = y \Delta_\alpha x^t.$$

Si se considera el conjunto fundamental de patrones $\{(x^\mu, y^\mu) \,|\mu = 1, 2, ..., p\}$ entonces la ij-esima entrada de la matriz $y^\mu \oplus (x^\mu)^t$ es expresada como:

$$\left[y^\mu \oplus (x^\mu)^t\right]_{ij} = \alpha(y_i^\mu, x_j^\mu).$$

2.1. Fase de aprendizaje

Encontrar los operadores adecuados y una manera de generar la matriz M esta guardará las p asociaciones del conjunto fundamental $\left\{\left(x^1, y^1\right), \left(x^2, y^2\right), ..., \left(x^p, y^p\right)\right\}$, donde $x^\mu \in A^n$ y $y^\mu \in A^m$ $\forall \mu \in \{1, 2, ..., p\}$.

Paso 1. Para cada asociación de patrones fundamentales $\{(x^\mu, y^\mu) \,|\mu = 1, 2, ..., p\}$, generar p matrices de acuerdo a la siguiente regla:

$$\left[y^\mu \oplus (x^\mu)^t\right]_{mxn}$$

Paso 2. Para obtener la Memoria Alfa-Beta tipo MAX, aplicar el operador binario MAX ($\vee$) de acuerdo a la siguiente regla:

$$M = \vee_{\mu=1}^{p} \left[y^{\mu} \oplus (x^{\mu})^{t} \right]$$

Paso 3. Para obtener un Memoria Alfa-Beta tipo *min* aplicar el operador binario *min* ($\wedge$) de acuerdo a la siguiente regla:

$$W = \wedge_{\mu=1}^{p} \left[y^{\mu} \oplus (x^{\mu})^{t} \right]$$

En consecuencia, la ij-esima entrada de la Memoria Alfa-Beta de tipo MAX esta dada por la siguiente expresión:

$$\nu_{ij} = \vee_{\mu=1}^{p} \alpha(y_i^{\mu}, x_j^{\mu})$$

Análogamente, la ij-esima entrada de una Memoria Asociativa Alfa-Beta de tipo *min* esta dada por la siguiente expresión:

$$\psi_{ij} = \wedge_{\mu=1}^{p} \alpha(y_i^{\mu}, x_j^{\mu}).$$

2.2. Fase de recuperación

Encontrar los operadores adecuados y las condiciones suficientes para obtener un patrón fundamental de salida y^{μ} cuando una memoria M o una memoria W es operada con un patrón fundamental de entrada x^{μ}.

Paso 1. Un patrón x^{ω}, con $\omega \in \{1, 2, ..., p\}$, es presentado a la Memoria Asociativa Alfa-Beta, entonces, x^{ω} es recuperado de acuerdo a una de las siguiente reglas.

Memoria Asociativa Alfa-Beta de tipo MAX:

$$M \Delta_{\beta} x^{\omega} = \wedge_{j=1}^{n} \beta(\nu_{ij}, x_j^{\omega}) = \wedge_{j=1}^{n} \left\{ \left[\vee_{\mu=1}^{p} \alpha(y_i^{\mu}, x_j^{\mu}) \right], x_j^{\omega} \right\}$$

Memoria Asociativa Alfa-Beta de tipo *min*:

$$W \bigtriangledown_{\beta} x^{\omega} = \vee_{j=1}^{n} \beta(\psi_{ij}, x_j^{\omega}) = \vee_{j=1}^{n} \left\{ \left[\wedge_{\mu=1}^{p} \alpha(y_i^{\mu}, x_j^{\mu}) \right], x_j^{\omega} \right\}$$

Sin importar el tipo de Memoria Asociativa Alfa-Beta usado la fase de recuperación, un vector columna de dimensión n será obtenido.

3. DEAQ

El diccionario de especialidades agro-químicas es un libro editado año con año por PLM desde 1989, este libro contiene información de más de 1800 productos agro-químicos, este libro puede ser consultado por:

Productos
Usos
Cultivos
Ingredientes activos

En este libro, la información proporcionada por la empresa que fabrica el producto es recopilada y clasificada, para proporcionar un gran conjunto de datos. La página principal del DEAQ puede ser accesada en [12].

4. Conjuntos de datos de herbicidas

La base de datos relacional, se generó como una abstracción del DEAQ este puede ser consultado en [13]. Se tomaron las características principales de 40 herbicidas, creando 3473 instancias. Las características de estas instancias son tóxicidad, método de acción, ingrediente activo, periodo de reentrada, incompatibilidad, contraindicaciones y primeros auxilios.

El conjunto de datos tiene la estructura siguiente:

{"maleza", çultivo", "tóxicidad", "periodo de reentrada", "herbicida"}

Donde "maleza", çultivo, "tóxicidadτ "periodo de reentrada"son los principales atributos a ser analizados y "herbicida.^{es} la mejor opción por elegir.

Se tienen 68 malezas, 50 cultivos, 2 tipos de tóxicidad y 6 periodos de reentrada diferentes.

El conjunto de datos esta mapeado a la base de datos relacional por los índices, cada número representa un elemento diferente ya sea maleza, cultivo, tóxicidad o herbicida.

5. Detalles de implementación

El presente trabajo fue realizado haciendo uso de las Memorias Asociativas Alfa-Beta tipo *min*, autoasociativas, estas memorias fueron combinadas con el código Johnson-Möbius [14].

Esto debido a que las Memorias Asociativas Alfa-Beta clasifican patrones de entrada con ruido aditivo o sustractivo, pero estas no son tan eficientes con ruido combinado, pero si incluimos la codificación de Johnson-Möbius, las Memorias Asociativas Alfa-Beta serán buenas clasificando patrones con ruido combinado.

Este código tiene una propiedad importante, cuando un valor es codificado en Johnson-Möbius este código preserva el ruido aditivo o substractivo pero no mixto.

Para generar el código Johnson-Möbius de un número de cero a $n-1$, se requieren $\frac{n}{2}$ bits. Si modificamos el código Johnson-Möbius para solo usar los códigos entre cero y $\frac{n}{2}$, donde $\frac{n}{2}$ es el número máximo representado (denotado como nmax) e invertimos el orden de los bits de la forma siguiente. El bit más significante se convierte en el bit menos significante, y el menos significante se convierte en el más significante. Por lo tanto, para representar un número que va de cero a nmax, se necesitan nmax bits.

El algorítmo para generar un número codificado en Johnson-Möbius es:

1.- Seleccionar el mayor de los valores dados

2.- Escribir el número como una concatenación de bits con el valor 1

3.- Completar con bits de valor 0 de izquierda a derecha hasta alcanzar la longitud n.

Ejemplo:

Valores: 3,5,10,6,13

1.- Seleccionamos el mayor de los valores dados, en este caso es 13, n=13.

2.- Escribir el número como una concatenación de bits con el valor 1

3: 111
5: 11111
10: 1111111111
6: 111111
13: 1111111111111

3.- Completar con bits de valor 0 de izquierda a derecha hasta alcanzar la longitud n.

3: 0000000000111
5: 0000000011111
10: 0001111111111
6: 0000000111111
13: 1111111111111

Debido a características del desarrollo se diseñaron dos Memorias Asociativas Alfa-Beta tipo min autoasociativas, una estricta y otra flexible, estas dos memorias proporcionan un mejor rango de soluciones, las dos memorias fueron entrenadas usando el mismo conjunto fundamental, la diferencia entre estas dos memorias es que la Memoria Asociativa Alfa-Beta tipo min autoasociativa estricta utiliza el conjunto completo de características en la fase de aprendizaje y recuperación al contrario de la Memoria Asociativa Alfa-Beta tipo min autoasociativa flexible que utiliza un conjunto reducido de características en la fase de aprendizaje y recuperación .Usando la característica de una completa recuperación de los patrones de salida la Memoria Asociativa Alfa-Beta tipo min estricta nos puede dar un excelente resultado si el usuario selecciona un patrón del conjunto fundamental, pero cuando un patrón entrante no pertenece al conjunto fundamental, se utiliza la Memoria Asociativa Alfa-Beta tipo min flexible. La memoria asociativa flexible se realizó utilizando la técnica de selección de características que nos ofrece las memorias Alfa-Beta, con esto seleccionamos las características principales, que son maleza y cultivo.

Después de codificar el patrón de entrada con el código de Johnson-Möbius se utiliza la Memoria Asociativa Alfa-Beta tipo min estricta para buscar un buen resultado si se encuentra, se muestra la información del producto, si no, el patrón de entrada va a la Memoria Asociativa Alfa-Beta tipo min flexible donde se busca un mejor resultado.

6. Resultados numéricos

Ejemplo 4.1. Sea $p = 5$, $n = 4$, $m = 4$. Dados los patrones fundamentales $\{(x^\mu, y^\mu) \mid \mu = 1, 2, ..., p\}$, se obtiene una Memoria Asociativa Alfa-Beta. Las asociaciones fundamental serán escritas como: $\{(x^1, y^1), (x^2, y^2), ..., (x^5, y^5)\}$.

$$x^1 = \begin{pmatrix} 1 \\ 1 \\ 0 \\ 1 \end{pmatrix} \quad x^2 = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 1 \end{pmatrix} \quad x^3 = \begin{pmatrix} 0 \\ 1 \\ 1 \\ 1 \end{pmatrix} \quad x^4 = \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \end{pmatrix} \quad x^5 = \begin{pmatrix} 1 \\ 0 \\ 1 \\ 1 \end{pmatrix}$$

$$y^1 = \begin{pmatrix} 1 \\ 1 \\ 0 \\ 1 \end{pmatrix} \quad y^2 = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 1 \end{pmatrix} \quad y^3 = \begin{pmatrix} 0 \\ 1 \\ 1 \\ 1 \end{pmatrix} \quad y^4 = \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \end{pmatrix} \quad y^5 = \begin{pmatrix} 1 \\ 0 \\ 1 \\ 1 \end{pmatrix}$$

Fase de aprendizaje. Obtenemos las matrices correspondientes $M_1, M_2, \ldots, M_5$, de acuerdo al paso 1, indicado en la sección 2.1.

$$y^1 \oplus \left(x^1\right)^t = \begin{pmatrix} 1 \\ 1 \\ 0 \\ 1 \end{pmatrix} \oplus \begin{pmatrix} 1 & 1 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 1 & 2 & 1 \\ 1 & 1 & 2 & 1 \\ 0 & 0 & 1 & 0 \\ 1 & 1 & 2 & 1 \end{pmatrix}$$

$$y^2 \oplus \left(x^2\right)^t = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 1 \end{pmatrix} \oplus \begin{pmatrix} 1 & 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 2 & 2 & 1 \\ 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 \\ 1 & 2 & 2 & 1 \end{pmatrix}$$

$$\vdots$$

$$y^5 \oplus \left(x^5\right)^t = \begin{pmatrix} 1 \\ 0 \\ 1 \\ 1 \end{pmatrix} \oplus \begin{pmatrix} 1 & 0 & 1 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 2 & 1 & 1 \\ 0 & 1 & 0 & 0 \\ 1 & 2 & 1 & 1 \\ 1 & 2 & 1 & 1 \end{pmatrix}$$

De acuerdo al paso 2, una Memoria Asociativa Alfa-Beta de tipo MAX representada por M, es obtenida, análogamente de acuerdo al paso 3 una Memoria Asociativa Alfa-Beta de tipo mín represenada por W, es obtenida

$$M = \begin{pmatrix} 1 & 2 & 2 & 1 \\ 2 & 1 & 2 & 2 \\ 2 & 2 & 1 & 1 \\ 2 & 2 & 2 & 1 \end{pmatrix} \quad ; \quad W = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 \end{pmatrix}$$

Fase de recuperación. Obtener el correspondiente patrón de salida, realizando las operaciones $M \triangle_\beta x^\mu$, $\forall \mu \in \{1, 2, ..., p\}$ como fue declarado en la sección 2.2. Debido a las limitaciones de espacio, solo la fase de recuperación de las Memorias Alfa-Beta tipo MAX es mostrada.

$$M \triangle_\beta x^1 = \begin{pmatrix} 1 & 2 & 2 & 1 \\ 2 & 1 & 2 & 2 \\ 2 & 2 & 1 & 1 \\ 2 & 2 & 2 & 1 \end{pmatrix} \triangle_\beta \begin{pmatrix} 1 \\ 1 \\ 0 \\ 1 \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \\ 0 \\ 1 \end{pmatrix} = y^1$$

$$M \triangle_\beta x^2 = \begin{pmatrix} 1\,2\,2\,1 \\ 2\,1\,2\,2 \\ 2\,2\,1\,1 \\ 2\,2\,2\,1 \end{pmatrix} \triangle_\beta \begin{pmatrix} 1 \\ 0 \\ 0 \\ 1 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 1 \end{pmatrix} = y^2$$

$$M \triangle_\beta x^3 = \begin{pmatrix} 1\,2\,2\,1 \\ 2\,1\,2\,2 \\ 2\,2\,1\,1 \\ 2\,2\,2\,1 \end{pmatrix} \triangle_\beta \begin{pmatrix} 0 \\ 1 \\ 1 \\ 1 \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \\ 1 \\ 1 \end{pmatrix} = y^3$$

$$M \triangle_\beta x^4 = \begin{pmatrix} 1\,2\,2\,1 \\ 2\,1\,2\,2 \\ 2\,2\,1\,1 \\ 2\,2\,2\,1 \end{pmatrix} \triangle_\beta \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \end{pmatrix} = y^4$$

$$M \triangle_\beta x^5 = \begin{pmatrix} 1\,2\,2\,1 \\ 2\,1\,2\,2 \\ 2\,2\,1\,1 \\ 2\,2\,2\,1 \end{pmatrix} \triangle_\beta \begin{pmatrix} 1 \\ 0 \\ 1 \\ 1 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \\ 1 \\ 1 \end{pmatrix} = y^5$$

El lector fácilmente puede verificar que la fase de recuperación de Alfa-Beta de tipo *min* también puede recuperar perfectamente y por completo el conjunto fundamental de patrones.

7. Resultados experimentales

El conjuto de datos utilizado para las pruebas, contenia 5 razgos, 40 clases y 3473 instancias, fue probado en "WEKA 3: Data Mining Software in Java"[15]. WEKA es un programa open source bajo la licencia "GNU General Public License", disponible de forma gratuita en internet [16]. Todos los experimentos fueron realizados usando una computadora personal con un procesador AMD Phenom II X4 955 a 3.2 GHz, corriendo sobre un Windows 8 Profesional con 8GB de RAM. En todas las pruebas se aplicaron las mismas condiciones y esquemas de validación para cada experimento. La condición para la experimentación fue enseñar al algoritmo con un conjunto de patrones y después recuperar todos los patrones. En la Tabla 2, se compara la eficiencia que tuvieron diversos algoritmos recuperando la totalidad de patrones enseñados, cabe resaltar, que las Memorias Asociativas Alfa-Beta tuvieron una recuperación de patrones del 100 % en la etapa de validación.

8. Conclusiones e investigación en progreso

Usando Memorias Asociativas Alfa-Beta, se puede recuperar completamente el conjunto fundamental, esto nos brinda una gran oportunidad para desarrollar un conjunto de aplicaciones enfocadas a la agronomía. En este artículo se

Tabla 2. Eficiencia de recuperacin probado con weka.

Algoritmo	Instancias clasificadas correctamente	Porcentaje
IB1	2510	72.27 %
RandomForest	2505	72.13 %
BayesNet	2484	71.52 %
Bagging	2444	70.37 %
AtributedSelectedClassifier	2361	67.98 %
J48	2361	67.98 %
END	2342	67.43 %
REPTree	2322	66.86 %
RandomSubSpace	2299	66.20 %
BFTree	2296	66.11 %
DataNearBalancedND	2275	65.51 %
ClassBalancedND	2269	65.33 %
REPTree	2200	63.35 %
SimpleCart	2189	63.03 %
Memorias Asociativas Alfa-Beta	3473	100 %

mostró que las Memorias Asociativas pueden facilmente ser usadas en cualquier tipo de desarrollo, en este caso particular una aplicación agronómica.

Particularmente sobre los sistemas que emiten recomendaciones, estos son muy útiles para el campo mexicano, ya que un ingeniero agrónomo, difícilmente, domina un catálogo extenso de productos agro-químicos y posee información sobre las condiciones exactas de aplicación, el ingeniero agrónomo puede ayudarse de este tipo de sistemas, cuando desconozca en cierta medida, el mejor producto para una aplicación en determinado cultivo y maleza.

Un sistema basado en Memorias Asociativas Alfa-Beta también puede recomendar el mejor momento para la aplicación de un producto, considerando variables como la velocidad del viento, la altura de la maleza y la posibilidad de lluvia, modelando todas estas características en diferentes ejemplos y usando las Memorias Asociativas Alfa-Beta como clasificador.

Actualmente investigamos sobre como mejorar las memorias asociativas a través de la selección de características, esto para implementar las memorias asociativas en otros campos de la ciencia.

Agradecimientos. Los autores del presente artículo qusieran dar las gracias a las siguientes instituciones por su apoyo: Instituto Politécnico Nacional, México (CIDETEC, ESCOM, CGPI, PIFI, COFAA) , CONACyT y SNI.

Referencias

1. Y. Hashimoto: Applications of artificial neural networks and genetic algorithms to agricultural systems. Computers and electronics in agriculture, 18, 71-72 (1997)

2. Dimitrios Moshou, Els Vrindts, Bart De Ketelaere, Josse De Baerdemaeker, Herman Ramon: A neural network based plant classifier, Computers and electronics in agriculture, 31, 5–16 (2001)
3. M.J. Aikenhead, I.A. Dalgetty, C.E. Mullins, A.J.S McDonald, N.J.C. Stranchan: Weed and crop discrimination using image analysis and artificial intelligence methods, Computers and electronics in agriculture, Computers and electronics in agriculture, 39, 157–171 (2003)
4. T. Karimi, S.O. Prasher, R.M. Patel, S.H. Kim: Application of support vector machine technology for weed and nitrogen stress detection in corn, Computers and electronics in agriculture, 51, 99–109 (2006)
5. Juan Ignacio Arribas, Gonzalo V. Sánchez-Ferrero, Gonzalo Ruiz-Ruiz, Jaime Gómez-Gil: Leaf classification in sunflower crops by computer vision and neural networks, 78, 9–18 (2011)
6. Acevedo-Mosqueda, M. E., Yáñez-Márquez, C., & López-Yáñez, I.: Alpha–Beta bidirectional associative memories: theory and applications. Neural Processing Letters, 26 (2007)
7. Steinbuch, K.: Die Lernmatrix. Kybernetik 1(1):36–45 (1961)
8. Anderson, J.A.: A simple neural network generating an interactive memory. Math Biosci 14:197–220 (1972)
9. Kohonen, T.: Correlation Matrix Memories. IEEE Transactions on Computers, 21, 353–359 (1972)
10. Hopfield, J.J.: Neural networks and physical systems with emergent collective computational abilities. In: Proc Nat AcadSci 79:2554–2558 (1982)
11. Acevedo-Mosqueda, M. E., Yáñez-Márquez, C., & López-Yáñez, I.: A New Model of BAM: Alpha-Beta Bidirectional Associative Memories. Lecture Notes in Computer Science (LNCS), 4263, 286–295 (2006)
12. PLM. DEAQ Diccionario de especialidades Agroquimicas. (2014) [Online]. http://www.agroquimicos-organicosplm.com/inicio
13. PLM. DEAQ Diccionario de especialidades Agroquimicas. (2014) [Online]. http://www.elcamporadio.com/source/
14. C. Yáñez, E.M.F. Riverón, I. López-Yáñez, R. Flores-Carapia: A novel approach to automatic color matching. In: CIARP, pp. 529–538 (2006)
15. M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, I.H. Witten: The WEKA data mining software: an update, SIGKDD Explorations 11 (1), pp. 10–18 (2009)
16. M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, I.H. Witten: WEKA 3: Data mining software in java (2010). URL http://www.cs.waikato.ac.nz/ml/weka/

Extensión de una red neuronal relacional difusa incorporando distintos productos relacionales a la etapa de entrenamiento

Efraín Mendoza Castañeda, Carlos Alberto Reyes García, Hugo Jair Escalante

Department of Computer Science,
Instituto Nacional de Astrofísica, Óptica y Electrónica (INAOE),
Puebla, México

{efrain.mendoza.c,kargaxxi,hugojair}@inaoep.mx

Resumen. En este trabajo se extiende la red neuronal relacional difusa basada en el modelo de Pedrycz. Dicha ampliación consiste en dotarla de la posibilidad de cambiar el producto relacional en la fase de entrenamiento. Los productos relacionales propuestos para esto son los llamados *BK-Products*: *SubTriangle*, *SupTriangle* y *Square*, además del uso de operadores más generales (*t-norms* y *s-norms*) en sus definiciones, esto también se aplica al producto relacional Circulo usado por Pedrycz. Exploramos la efectividad de esta extensión en problemas de clasificación (incluyendo bases de datos con un esquema de ruido) y encontramos que en muchos casos la habilidad de clasificación de esta red es incrementada.

Palabras clave: Redes neuronales relacionales difusas, computo suave, sistemas neuro-difusos, productos relacionales.

1. Introducción

Los sistemas neuronales y difusos se combinan naturalmente asemejando a un sistema adaptativo con componentes sensoriales y cognitivos [15]; a los métodos que generan esta combinación se les conoce como métodos de hibridación neuro-fuzzy, estos pueden ser agrupados en dos grandes grupos [13]: (I) como redes neuro-difusas (*Fuzzy Neural Networks, FNN*), donde una red neuronal es equipada con la capacidad para manejar información difusa; y (II) como sistemas difuso-neurales (*Neuro-fuzzy systems, NFS*) que involucran un sistema de inferencia difuso (*Fuzzy Inference System, FIS*) combinado con una red neuronal para proporcionarle capacidad de aprendizaje. El presente trabajo se centra en el primer método de hibridación, donde las neuronas son diseñadas para ejecutar varias operaciones usadas en la teoría de conjuntos difusos en lugar de las operaciones comunes de multiplicación y adición, concretamente sobre las llamadas redes neuronales relacionales difusas (*Fuzzy Relational Neural Network, FRNN*).

La estructura del presente trabajo queda como sigue: en la Sección 2 se da detalle de la forma y funcionamiento de la FRNN usada, la extensión propuesta

se muestra en Sección 4, las pruebas para examinar el comportamiento de la red son establecidas en la Sección 5 (la FRNN es enfrentada a problemas de clasificación), finalmente, las secciones 6 y 7 muestran un análisis de los resultados obtenidos.

2. Red neuronal relacional difusa (FRNN)

Una FRNN usa productos relacionales (*Relationals Products*, RP), también llamados composiciones, para su funcionamiento. Un RP es un operador binario que opera sobre una relación R, entre un conjunto X y un conjunto Y, y una relación S, de Y a Z, cuyo resultado es una relación del conjunto X al conjunto Z, esto es:

$$[\Re(X \to Y) \times \Re(Y \to Z)] \to \Re(X \to Z) \tag{1}$$

estos RPs están compuestos por operaciones difusas: intersección, unión e implicación. Las operaciones de intersección y unión sobre conjuntos difusos son generalizadas por medio de las llamadas *t-norms* $\wedge$ y *s-norms* $\vee$ [8], respectivamente.

En las siguientes subsecciones se explica con más detalle el funcionamiento del modelo de FRNN utilizado.

2.1. Red de Pedrycz

Un perceptrón *feed-forward* de una capa esta formado por una colección de nodos de entrada $X = \{x_1, x_2, ..., x_m\}$, una colección de nodos de salida $Y = \{y_1, y_2, ..., y_l\}$, y una matriz de pesos $W = \{w_{ij} | i \in m, j \in l\}$, donde l son las clases a las que puede pertenecer el patrón de entrada. La salida del nodo $y_j = f(\sum_i x_i w_{ij})$ es una función de la suma ponderada por los pesos de las entradas. Un nodo de sesgo (*bias*) x_0 puede ser incluido.

Pedrycz [14] reemplaza estos componentes por relaciones difusas. La matriz de pesos del perceptrón es reemplazada por una matriz relacional difusa que representa una relación difusa de X a Y, $R = \{xRy | x \in X, y \in Y\}$, de tal forma que la conexión entre x_i y y_j tiene un valor relacional $R(x_i, y_j)$. Los valores de salida Y son generados por el producto relacional del conjunto de entradas X y las relaciones R, es decir $Y = X \circ R$. Para esta red Pedrycz utiliza la composición *max-min* [17]. Esta composición se representa como $y_j = max(x_i, min(x_i, R(x_i, y_j)))$.

En [14] se propone también un índice de igualdad difuso basado en la implicación de Łukasiewicz, esto provee de una medida de desempeño para la evaluación del error en el entrenamiento. Esta métrica puede ser usada en técnicas de gradiente descendente para actualizar los pesos de la matriz en una forma similar al algoritmo estándar de retropropagación (*Backpropagation*, BP). Puesto que las técnicas de gradiente descendente requieren que la función usada sea derivable,

Pedrycz calcula un aproximado de la derivada de la composición *max-min*. Muestra también, que el problema XOR, un problema irresoluble por las redes estándar de una capa tiene solución y converge con la red propuesta.

2.2. Modificaciones de Reyes-García

Reyes-García propone (ver [15]) un marco de trabajo para explotar el uso de la FRNN de Pedrycz, con la particularidad de que esta FRNN usa diferentes productos relacionales en la etapa de procesamiento (descrita en breve). En esta arquitectura la capa de entrada esta formada por $N \times n$ neuronas, cada una de las cuales corresponde a una de los N términos lingüisticos asignados a cada una de las n entradas. La capa de salida esta compuesta por l neuronas, cada una de las cuales pertenece a una de las l clases. Existe una conexión entre cada nodo en la capa de entrada a cada nodo en la capa de salida. La Figura 1 muestra el marco de trabajo y las fases de la FRNN. Cada fase y cada módulo serán descritos posteriormente. La operación de la FRNN es dividida en dos fases; la primera es para aprendizaje y la segunda para el procesamiento.

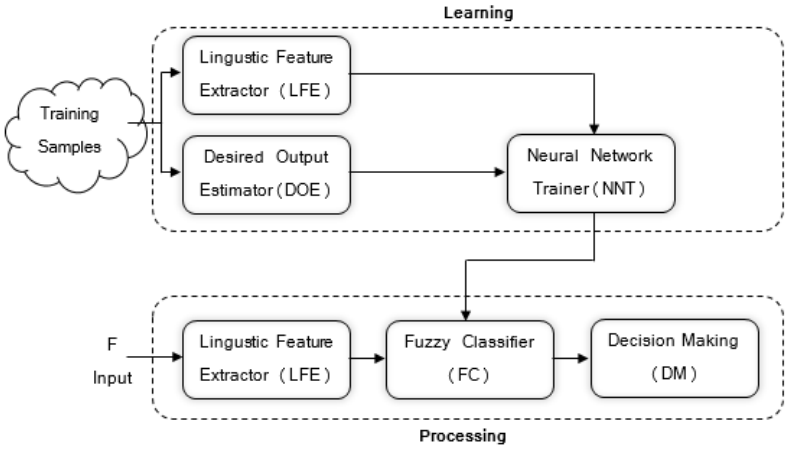


Fig. 1. Diagrama de bloques del marco de trabajo de un sistema basado en una Red Neuronal Relacional Difusa (FRNN).

2.2.1. Fase de aprendizaje

La fase de aprendizaje esta dividida en tres módulos: el Extractor de Características Lingüisticas (*Linguistic Feature Extractor*, LFE), Estimador de Salidas Deseadas (*Desired Output Estimator*, DOE) y el Entrenador de la Red Neuronal (*Neural Network Trainer*, NNT).

módulo LFE. Toma las muestras de entrenamiento F y cada característica de F es transformada en valores de membresía a cada uno de las propiedades lingüisticas asignadas. De este modo un vector conteniendo n características es transformado en un 3n-dimensional (describiendo pertenencia baja, media y alta), 5n-dimensional (pertenencia muy baja, baja, media, alta y muy alta), o 7n-dimensional (pertenencia muy baja, baja, mas o menos baja, media, más o menos alta, alta, muy alta). El vector resultante es llamado Vector de Propiedades Lingüisticas (*Linguistic Properties Vector*, LPV). Para calcular los valores de membresía pueden ser usadas diferentes tipos de funciones tales como: *Gaussian*, Trapezoidal, Triangular y *Bell* (ver [16]).

módulo DOE. Dado que, una FRNN es un método de aprendizaje supervisado, el segundo módulo, DOE, se encarga de calcular los valores de membresía para cada ejemplo en cada clase de salida, el arreglo resultante es llamado Vector de Salida Deseado (*Desired Output Vector*, DOV), que es posteriormente utilizado, en el calculo del error de la red después de cada iteración. Para obtener los valores de membresía deseados es necesario calcular la distancia del patrón de entrenamiento F_i a la $k - ésima$ clase como en la ecuación:

$$z_{ik} = \sqrt{\sum_{j=0}^{n} \left[\frac{F_{ij} - \mu_{kj}}{\sigma_{kj}} \right]^2} | k = 1...l \tag{2}$$

donde F_{ij} es la $j - ésima$ característica del patrón i, l el número de clases, μ_{kj} y σ_{kj} denotan, respectivamente, la media y la desviación estándar de la característica j y la clase k. El valor de membresía de F_i^{th} es definido como sigue:

$$\mu_l(F_i) = \frac{1}{1 + \left(\frac{z_{ik}}{f_d} \right)^{f_e}} \tag{3}$$

donde f_e es el generador exponencial difuso, y f_d es el generador denominacional difuso, estos parámetros controlan la cantidad de difusión en esta clase. En este caso, mientras mayor sea la distancia del patrón a una clase, menor será su membresía a esta. Dado que los datos de entrenamiento tienen límites de clase difusos, un patrón puede pertenecer a una o varias clases al mismo tiempo en diferentes grados.

módulo NNT. Toma los vectores LPV y DOV como la base para entrenar la red. El LPV se pasa a la capa de entrada y el DOV es utilizado para el entrenamiento de la red. Las salidas de la red son calculadas para obtener el error de la capa de salida. El error esta representado por la distancia entre la salida actual y la salida objetivo. Minimizar este error es el objetivo del proceso de entrenamiento. Durante cada paso del aprendizaje, una vez que el error ha sido calculado, el entrenador ajusta los valores de la matriz de relaciones de las conexiones correspondientes, por medio el algoritmo BP, hasta que se obtiene un error mínimo o se ha completado un número dado de iteraciones. La salida

del NNT es una matriz relacional que contiene el conocimiento necesario para posteriormente mapear un vector desconocido de entrada a su correspondiente clase durante la fase de procesamiento. El proceso de aprendizaje es explicado en detalle en [14].

2.2.2. Fase de procesamiento

La fase de procesamiento comienza usando el mismo LFE definido en la fase de aprendizaje, en esta fase el LFE opera en una forma similar para *fuzzificar* los datos de prueba. El clasificador difuso (FC, *Fuzzy Classifier*) entonces usa la matriz relacional desarrollada durante la fase de aprendizaje y el LPV para procesar los patrones de entrada. Los patrones de entrada son procesados usando diferentes RPs. El último módulo, DMM, es el encargado de interpretar la salida difusa de la FRNN y depende del problema bajo análisis, para fines de clasificación toma la membresía más alta del vector de clases y la asigna a la muestra de entrada.

3. Trabajo relacionado

La composición *max-min* es una de las más usadas ([5–7, 12, 14, 15]). W. Pedrycz, como se mencionó anteriormente, [14] propone la estructura básica de la FRNN. Blanco et al. [5] integra el concepto de "derivación suave" a la aproximación de la derivada; Valente de Oliveira [12] expande la composición a *max-t*. Reyes-García [15] extiende la red desarrollada por Pedrycz [14] incorporando los productos relacionales *BK* [2]- [9]; Davis & Kohout [6] incorporan generalizaciones de los RPs *BK*; Barajas y Reyes (ver [4]) usan un algoritmo genético (para seleccionar número de términos lingüísticos, tipo de funciones de membresía, método de clasificación y tasa de aprendizaje) en combinación con el método BP. Otros trabajos ([1]) se centran en el uso de normas difusas suaves (*smooth fuzzy norms*), es decir, normas que son derivables.

En el presente trabajo se extienden los métodos presentados por [15] y [6]. La extensión introducida consiste en la integración de otros productos relacionales en la etapa de entrenamiento.

4. Extensión propuesta

Este trabajo propone utilizar diferentes RPs en la etapa de entrenamiento, así como el uso de *t-norms* y *s-norms* en las definiciones de estos.

La incorporación de más RPs a la FRNN se hace con base en las observaciones hechas por Bandler & Kohout [3]:

- Existen muchas maneras significativas distintas de definir la inclusión de una estructura difusa en otra; esto depende de la elección de un operador de implicación en particular.

- La elección de las propiedades semánticas de un operador de implicación particular depende de algunas consideraciones pragmáticas, determinadas por las preguntas metodológicas de la aplicación en particular.

De lo anterior se desprende que, si solo entrenamos a la FRNN con una composición (*max-min* en este caso), limitamos la habilidad de esta para representar relaciones entre estructuras, por lo que se hace necesario proveerla de la capacidad de sustituir el RP de entrenamiento por otro que resulte más conveniente, dependiendo del problema bajo análisis.

En las siguientes subsecciones se muestra cual es el camino tomado para incorporar otros productos relacionales al entrenamiento de la FRNN.

4.1. Fase de entrenamiento

El cambio de un RP por otro en el entrenamiento de la FRNN no puede hacerse de forma transparente; puesto que el algoritmo de aprendizaje (BP) usado por la FRNN es un método de gradiente descendente y requiere que los operadores utilizados para el entrenamiento sean derivables, condición que la mayoría de los operadores difusos no cumple. En este punto se tienen dos opciones: (1) limitarse al uso de operadores que sean derivables (2) calcular para los operadores - en caso de que no sean derivables - un aproximado de su derivada. Nosotros nos apegaremos a la segunda opción, puesto que existen muchos operadores que han probado ser de gran valor al representar las relaciones existentes entre patrones de datos.

Pedrycz incorpora un índice de igualdad Q expresado por implicaciones difusas (ver [14]), este índice sirve como medida para evaluar el error de la red, al calcular la derivada de este error respecto a los pesos, obtiene lo siguiente:

$$\Delta w = \frac{\partial Q}{\partial w} = -\sum_{i:y_i>t_i} \frac{\partial f(x_i; w, \vartheta)}{\partial w} + \sum_{i:y_i<t_i} \frac{\partial f(x_i; w, \vartheta)}{\partial w} \tag{4}$$

$$\Delta \delta = \frac{\partial Q}{\partial \delta} = -\sum_{i:y_i>t_i} \frac{\partial f(x_i; w, \vartheta)}{\partial \delta} + \sum_{i:y_i<t_i} \frac{\partial f(x_i; w, \vartheta)}{\partial \delta} \tag{5}$$

donde x_i es el patrón de entrada, w es la matriz de relaciones y ϑ es el vector de sesgo. El resto de la derivada (Δw o $\Delta \delta$) puede ser calculado cuando se especifica la forma de la función f. Por ejemplo, si tomamos en cuenta la definición del producto *circlet* ($\circ$) con un sesgo, $y_k = \vee\left(\vee\left(x_{ij} \wedge w_{kj}\right), \vartheta_k\right)$ donde $k = 1, .., l$, tenemos (los índices son omitidos para mayor claridad):

$$\begin{aligned} \frac{\partial(\vee(w \wedge x) \vee \vartheta)}{\partial w} &= \frac{\partial(\vee(w \wedge x) \vee \vartheta)}{\partial \vee (a \wedge x)} * \frac{\partial(\vee(w \wedge x))}{\partial w} \\ &= \frac{\partial(\vee(w \wedge x) \vee \vartheta)}{\partial \vee (w \wedge x)} * \frac{\partial(\vee(w \wedge x))}{\partial(w \wedge x)} * \frac{\partial(w \wedge x)}{\partial w} \end{aligned} \tag{6}$$

esta derivada esta en términos de *t-norms* y *s-norms*, estos operadores tienen que ser sustituidos por implementaciones que cumplan con las restricciones inherentes a estos. De este modo si reemplazamos las *t-norms* por el operador *min* y las *s-norms* por *max* (considerando que conocemos las aproximaciones de sus derivadas [14] [5]), la derivada queda como sigue

$$\begin{aligned}
\frac{\partial(\vee(w \wedge x) \vee \vartheta)}{\partial \vee (w \wedge x)} &= \begin{cases} 1 & \vee(w \wedge x) \geq \vartheta, \\ 0 & otro\ caso. \end{cases} \\
\frac{\partial(\vee(w \wedge x))}{\partial(w \wedge x)} &= \begin{cases} 1 & (w \wedge x) \geq (\vee(w \wedge x), \\ 0 & otro\ caso. \end{cases} \\
\frac{\partial(w \wedge x)}{\partial w} &= \begin{cases} 1 & a \leq x, \\ 0 & otro\ caso. \end{cases}
\end{aligned} \tag{7}$$

como se puede observar la derivada del producto $max - min$ se reduce a una serie de casos, que son iguales a los mostrados por Pedrycz [14]. Adoptando este enfoque podemos agregar más RPs para el entrenamiento definiendo la derivada del producto relacional hasta el punto que se muestra en la ecuación (6) y posteriormente, el resto del proceso es transparente.

Los RPs que en este trabajo se integrarán a la fase de entrenamiento son denominados *BK products* (por Bandler-Kohout): SupTriangle, SubTriangle y Square; por las cualidades de estos para representar relaciones entre estructuras [10]. Las derivadas y definiciones de estos productos se muestran en la Tabla 1.

Tabla 1. Productos relacionales que se incorporarán a la etapa de entrenamiento.

Producto Relacional.	Definicion	Derivada
Circlet ($\circ$)	$\vee_j(R_{ij} \wedge S_{ij})$	$\frac{\partial(\vee(w\wedge x)\vee\vartheta)}{\partial\vee(w\wedge x)} * \frac{\partial(\vee(w\wedge x))}{\partial(w\wedge x)} * \frac{\partial(w\wedge x)}{\partial w}$
SubTriangle ($\triangleright$)	$\wedge_j(R_{ij} \rightarrow S_{jk})$	$\frac{\partial(\wedge(x\rightarrow w)\vee\vartheta)}{\partial(\wedge(x\rightarrow w))} * \frac{\partial(\wedge(x\rightarrow w))}{\partial(x\rightarrow w)} * \frac{\partial(x\rightarrow w)}{\partial w}$
SupTriangle ($\triangleleft$)	$\wedge_j(R_{ij} \leftarrow S_{jk})$	$\frac{\partial(\wedge(x\leftarrow w)\vee\vartheta)}{\partial(\wedge(x\leftarrow w))} * \frac{\partial(\wedge(x\leftarrow w))}{\partial(x\leftarrow w)} * \frac{\partial(x\leftarrow w)}{\partial w}$
Square ($\square$)	$\wedge_j(R_{ij} \leftrightarrow S_{jk})$	$\frac{\partial(\wedge(x\leftrightarrow w)\vee\vartheta)}{\partial(\wedge(x\leftrightarrow w))} * \frac{\partial(\wedge(x\leftrightarrow w))}{\partial(x\leftrightarrow w)} * \frac{\partial(x\leftrightarrow w)}{\partial w}$

Para completar el proceso de entrenamiento necesitamos definir las *t-norms*, *s-norms* e implicaciones, con sus respectivas derivadas, que los RPs utilizarán. Los operadores fueron elegidos por su amplio uso en la literatura, los detalles de estos son mostrados en la Tabla 2.

Tabla 2. Operadores difusos usados por los RPs en la etapa de entrenamiento.

Operador	Implementación		Derivada
$\rightarrow$	Łukasewicz	$min(1, 1-a+b)$	$-\frac{\partial min(1,1-b+a)}{\partial a}$
$\leftrightarrow$		$min(a \rightarrow b, b \rightarrow a)$	$\frac{\partial min(a\rightarrow b, b\rightarrow a)}{\partial a} = \frac{\partial min(a\rightarrow b, b\rightarrow a)}{\partial a \rightarrow b} * \frac{\partial a \rightarrow b}{\partial a} + \frac{\partial min(a\rightarrow b, b\rightarrow a)}{\partial b \rightarrow} * \frac{\partial b \rightarrow a}{\partial a}$
$\vee$	Máximo	$max(a, b)$	$\frac{\partial max(a,b)}{\partial a} = \begin{cases} 1, & a >= b \\ 0, & otros \end{cases}$
$\wedge$	Mínimo	$min(a, b)$	$\frac{\partial min(a,b)}{\partial a} = \begin{cases} 1, & a <= b \\ 0, & otros \end{cases}$

4.2. Fase de procesamiento

Como se describió anteriormente en los trabajos de Reyes-García y Davis & Kohout, la FRNN ha sido extendida usando distintos RPs en la etapa de procesamiento de la red. En el presente trabajo nos apegaremos a este enfoque: usaremos como base para procesamiento los RPs *Circlet*, *Suptriangle*, *Subtriangle* y *Square* (ver Tabla 3). Los operadores difusos usados como instancias particulares de las *t-norm*, *s-norm* e implicación se muestran en Tabla 3.

5. Experimentos

Para probar la validez de la extensión propuesta la FRNN fue sometida a problemas de clasificación, esto es, asignar etiquetas de clase l, de un conjunto de etiquetas L, a los ejemplos de prueba, donde los valores de las características son conocidos, pero la etiqueta de clase no [11]. Una FRNN puede utilizarse para afrontar este problema gracias a la interpretación que puede darse a las salidas de la red. Estas salidas son grados de pertenencia a cada clase de $l \in L$, para problemas de clasificación el módulo DM asignará la etiqueta con mayor grado de pertenencia al patrón de entrada.

La evaluación fue realizada usando una validación cruzada de 10 pliegues y el criterio para la evaluación fue el porcentaje de patrones clasificados correctamente (exactitud).

5.1. Conjuntos de datos

Los conjuntos de datos utilizados se obtuvieron del *UCI Machine Learning Repository*: *iris*, *car*, *ecoli*, *pima*, *nursery*, *glass*, *wine*, *heart*, *ionosphere* (un resumen de las propiedades de los conjuntos de datos es dado en la Tabla 4). Una de las fortalezas de los algoritmos que utilizan lógica difusa es la tolerancia al ruido, por lo que en las pruebas de clasificación se toman en cuenta algunas

Tabla 3. Productos relacionales que se incorporarán a la etapa de procesamiento.

	Operador	Definición
Implicación $a \rightarrow b$	Łukasewicz	$min(1, 1 - a + b)$
	Kleene-Diens	$\vee(1 - a, b)$
	Gaines	$\wedge(1, b/a)$
Equidad $a \leftrightarrow b$		$\wedge(a \rightarrow b, b \rightarrow a)$
t-norm $\wedge(a, b)$	Mínimo	$min(a, b)$
	Producto	$a * b$
	Einsten product	$\frac{a*b}{2-a+b-a*b}$
	Hamacher product	$\frac{a*b}{a+b-a*b}$
	Drastic *t-norm*	$\begin{cases} b \text{ if } a == 1 \\ a \text{ if } b == 1 \\ 0 \text{ otros casos} \end{cases}$
	Nilpotent	$\begin{cases} min(a, b) \text{ if } (a + b) > 1 \\ 0 \text{ otros casos} \end{cases}$
s-norm $\vee(a, b)$	Máximo	$max(a, b)$
	Probabilistic sum	$a + b - a * b$
	Einsten product	$\frac{a+b}{1+a*b}$
	Hamacher sum	$\frac{a+b-2*(a*b)}{1-a*b}$
	Drastic *s-norm*	$\begin{cases} b \text{ if } a == 0 \\ a \text{ if } b == 0 \\ 1 \text{ otros casos} \end{cases}$
	Nilpotent	$\begin{cases} max(a, b) \text{ if } (a + b) < 1 \\ 1 \text{ otros casos} \end{cases}$

bases de datos con ruido. Las bases de datos fueron tomadas del *Keel-dataset repository* y el ruido fue introducido con el patrón propuesto por Wu et al [18] con un esquema *Noisy Train - Noisy Test* al 20 % (información más detallada de las bases de datos tomadas se muestran en la Tabla 4, marcadas con un '*').

Tabla 4. Estadísticas de las bases de datos.

Bd	#Atributos (R/I/N)	# Ejemplos	# Clases
*iris	4 (4/0/0)	150	3
car	6 (0/0/6)	1728	4
ecoli	7 (7/0/0)	336	8
*pima	8 (8/0/0)	768	2
glass	9 (9/0/0)	214	7
*wine	13 (13/0/0)	178	3
*heart	13 (1/12/0)	270	2
ionosphere	33 (32/1/0)	351	2

5.2. Configuración de la FRNN

La FRNN requiere de los siguientes parámetros para funcionar: RP para procesamiento, RP para entrenamiento, número de términos lingüísticos, tipo

de función de membresía y número de épocas. Los RPs para el procesamiento son el resultado de todas las posibles combinaciones que se originan de sustituir los operadores (Tabla 3) en la definición general de los RPs (Tabla 1). Para el entrenamiento de la red se tomaron los RPs listados en Tabla 1 combinados con los operadores Tabla 2. El número de términos lingüísticos puede establecerse en 3, 5 y 7 (ver detalle en [15]). Las funciones de membresía son usadas para calcular el nivel de pertenencia de un patrón de entrada a una clase, para este trabajo las elecciones posibles de funciones de membresía son *Pi*, *Triangular* y *Trapezoidal*. El número de épocas fue establecido en 5 ya que entrenamiento adicional no produjo efectos significativos en el rendimiento sobre los conjuntos de pruebas.

Todas las configuraciones de FRNN posibles, con los parámetros fijados anteriormente, fueron probadas sobre cada una de las bases de datos.

6. Resultados

Como se puede ver en la Tabla 5 los resultados obtenidos con la extensión propuesta en este trabajo mejoran a los alcanzados por la red de Pedrycz y la extensión de Reyes-García & Bandler.

Tabla 5. Comparación de desempeño de la red FRNN original y la extensión propuesta en este artículo. La segunda sección muestra los resultados obtenidos sobre las bases de datos con ruido.

Conjunto de datos	FRNN	E. FRNN
iris	96.66 ± 3.51	97.33 ± 3.44
car	78.47 ± 1.57	79.39 ± 2.73
ecoli	65.79 ± 4.03	73.25 ± 8.24
pima	74.35 ± 3.16	74.35 ± 3.16
glass	59.87 ± 6.09	62.34 ± 6.04
wine	87.09 ± 5.92	97.22 ± 3.92
heart	76.29 ± 8.76	84.81 ± 8.63
ionosphere	88.04 ± 4.56	88.04 ± 4.00
iris	82.66 ± 5.62	90.66 ± 7.82
pima	71.22 ± 3.29	72.13 ± 4.56
wine	80.35 ± 8.10	89.86 ± 81.44
heart	72.96 ± 7.20	81.85 ± 74.55

Es importante resaltar que el mejor RP de entrenamiento presentó variación dependiendo del conjunto de datos procesado, pues este era el resultado esperado; esto puede observarse en la Tabla 6, que muestra las mejores configuraciones de FRNN encontradas.

Tabla 6. Configuraciones de FRNN con mejores resultados (exactitud). La segunda sección de la tabla muestra las mejores configuraciones para las bases de datos con ruido.

Dataset	Función de membresía	# térm. ling.	RP Entrenamiento | RP procesamiento	Precisión
iris	Triangular	3	∘{∧ :Min, ∨ :Max} ◁{∧ :Min, → {KleeneDiens{∧ :HamacherSum}}}	97.33 ± 3.44
car	Trapezoidal	7	∘{∧ :Min, ∨ :Max} ◁{∧ :Min, → {KleeneDiens{∧ :HamacherSum}}}	9.39 ± 2.73
ecoli	Trapezoidal	7	□{↔ {Lukasewicz{∧ :Min}}, ∧ :Min} ∘{∧ :Ham.Prod.}, ∨ :ProbSum}	73.25 ± 8.24
pima	Triangular	3	∘{∧ :Min, ∨ :Max} ∘{∧ :Min, ∨ :Max}	74.35 ± 3.16
glass	Triangular	3	∘{∧ :Min, ∨ :Max} ∘{∧ :Nilp, ∨ :Max}	62.34 ± 6.04
wine	Pi	5	□{↔ {Lukasewicz{∧ :Min}}, ∧ :Min} ∘{∧ :HamacherProd, ∨ :EinstenSum}	97.22 ± 3.92
heart	Pi	5	◁{∧ :Min, → {Lukasewicz{∧ :Min}}} ∘{∧ :HamacherProd, ∨ :HamacherSum}	84.81 ± 8.63
ionosphere	Trapezoidal	5	□{↔ {Lukasewicz{∧ :Min}}, ∧ :Min} ∘{∧ :Min, ∨ :EinstenProduct}	88.04 ± 4.00
iris	Pi	3	▷{∧ :Min, → {Lukasewicz{∧ :Min}}} ∘{∧ :Min, ∨ :Prob.Sum}	90.66 ± 7.82
pima	Trapezoidal	7	∘{∧ :Min, ∨ :Max} ∘{∧ :Min, ∨ :SNilp.}	72.13 ± 4.56
wine	Trapezoidal	5	□{↔ {Lukasewicz{∧ :Min}}, ∧ :Min} □{↔ {→ {KD{∨ :Max}}, ∧ :Min}, ∧ :Product}	89.86 ± 81.44
heart	Triangular	7	◁{∧ :Min, → {Lukasewicz{∧ :Min}}} ∘{∧ :Ham.Prod.}, ∨ :ProbSum}	81.85 ± 74.55

[1] La descripción del RP seleccionado (para entrenamiento o procesamiento) se da en formato *JSON*, este formato esta formado de un conjunto desordenado de pares nombre/valor. Un objeto comienza con '{' (llave de apertura) y termine con '}' (llave de cierre). Cada nombre es seguido por ':' (dos puntos) y los pares nombre/valor están separados por ',' (coma).

7. Conclusiones y trabajo futuro

En este trabajo se exploró la incorporación de otros RPs a la etapa de entrenamiento de una FRNN. Se plantea una forma sencilla de agregar, a conveniencia, otros RPs más adecuados a la aplicación. Encontramos que, para algunas bases de datos, entrenando la FRNN con los RPs propuestos (*Circlet* generalizado), *SubTriangle*, *SupTriangle* y *Square*) se consiguen resultados que superan a la composición usada por Pedrycz (*max-min*).

La principal desventaja de este enfoque, es que, al agregar un mayor número de operadores puede ser prohibitivo evaluar todas las posibles combinaciones, por lo que, como trabajo futuro se plantea la adición de un método de selección de modelo que encuentre la mejor configuración de FRNN para la base de datos bajo análisis; esto con la finalidad de mitigar el inconveniente del rápido crecimiento de combinaciones posibles de FRNN. También, teniendo en cuenta que las salidas de la FRNN representan niveles de membresía del patrón procesado a cada una de las clases, se usará esta aproximación a problemas que den un mayor uso a esta información, como *Label ranking*.

Referencias

1. Ashtiani, A.A., Menhaj, M.B.: Numerical solution of fuzzy relational equations based on smooth fuzzy norms. Soft Computing 14(6), 545–557 (2010)
2. Bandler, W., Kohout, L.J.: Mathematical relations, their products and generalized morphisms. Techn Report Man-Machine Systems Lab Dept Electrical Engin Univ Essex, Colchester, Essex, UK, EES-MMS-REL pp. 77–3 (1977)
3. Bandler, W., Kohout, L.J.: Semantics of implication operators and fuzzy relational products. International Journal of Man-Machine Studies 12(1), 89–116 (1980)
4. Barajas, S.E., Reyes, C.A.: Your fuzzy relational neural network parameters optimization with a genetic algorithm. In: Fuzzy Systems, 2005. FUZZ'05. The 14th IEEE International Conference on. pp. 684–689. IEEE (2005)
5. Blanco, A., Delgado, M., Requena, I.: Identification of fuzzy relational equations by fuzzy neural networks. Fuzzy Sets and Systems 71(2), 215–226 (1995)
6. Davis, I., Warren, L.: Enhancing pattern classification with relational fuzzy neural networks and square bk-products (2006)
7. Garcia, C.A.R., Bandler, W.: Implementing a fuzzy relational neural network for phonetic automatic speech recognition. In: Fuzzy Modelling, pp. 115–139. Springer (1996)
8. Klement, E.P., Mesiar, R., Pap, E.: Triangular norms. position paper i: basic analytical and algebraic properties. Fuzzy Sets and Systems 143(1), 5–26 (2004)
9. Kohout, L.J.: Boolean and fuzzy relationsboolean and fuzzy relations. In: Encyclopedia of Optimization, pp. 189–202. Springer (2001)
10. Kohout, L.J., Kim, E.: The role of bk-products of relations in soft computing. Soft Computing 6(2), 92–115 (2002)
11. Kotsiantis, S.B.: Supervised machine learning: a review of classification techniques. Informatica (03505596) 31(3) (2007)
12. de Oliveira, J.: Neuron inspired learning rules for fuzzy relational structures. Fuzzy sets and systems 57(1), 41–53 (1993)
13. Pal, S.K., Mitra, S.: Neuro-fuzzy pattern recognition: methods in soft computing. John Wiley & Sons, Inc. (1999)
14. Pedrycz, W.: Neurocomputations in relational systems. IEEE Transactions on Pattern Analysis and Machine Intelligence 13(3), 289–297 (1991)
15. Reyes, C.A.: On the design of a fuzzy relational neural network for automatic speech recognition. Ph.D. thesis, Doctoral Dissertation, The Florida State University, Tallahassee, Fl (1994)
16. Rosales-Pérez, A., Reyes-García, C.A., Gómez-Gil, P.: Genetic fuzzy relational neural network for infant cry classification. In: Pattern Recognition, pp. 288–296. Springer (2011)
17. Zadeh, L.A.: Similarity relations and fuzzy orderings. Information sciences 3(2), 177–200 (1971)
18. Zhu, X., Wu, X., Yang, Y.: Error detection and impact-sensitive instance ranking in noisy datasets. In: AAAI. pp. 378–384 (2004)

Síntesis óptima de un mecanismo plano para seguimiento de trayectoria utilizando evolución diferencial

Eduardo Vega-Alvarado[1], Eric Santiago-Valentín[1], Alvaro Sánchez-Márquez[2], Adrián Solano-Palma[1], Edgar Alfredo Portilla-Flores[1], Leticia Flores-Pulido[2]

[1]Instituto Politécnico Nacional,
Centro de Innovación y Desarrollo Tecnológico en Cómputo,
México D.F., México

[2]Universidad Autónoma de Tlaxcala,
Facultad de Ciencias Básicas, Ingeniería y Tecnología,
Apizaco, Tlaxcala, México

{evega, asolanop, aportilla}@ipn.mx, asanchez@uat.mx,
{e.santiago.valentin, aicitel.flores}@gmail.com

Resumen. En este trabajo se presenta la síntesis de un mecanismo de cuatro barras para seguimiento de una trayectoria lineal de seis puntos. Para resolver el problema de optimización numérica con restricciones asociado al mecanismo se utiliza el algoritmo de evolución diferencial, modificado para aplicar una selección tipo torneo basada en las reglas de factibilidad de Deb. Los resultados alcanzados muestran una muy alta precisión en el seguimiento de la trayectoria propuesta, superando los valores mínimos obtenidos en otros trabajos similares.

Palabras clave: Síntesis, optimización, mecanismo de cuatro barras, evolución diferencial, restricciones.

1. Introducción

Uno de los mecanismos que más se utiliza en el diseño de maquinaria es el de cuatro barras, debido a que se ha comprobado que en sus diferentes configuraciones es el mecanismo articulado más simple para movimiento controlado de un grado de libertad [1].

Sin pérdida de generalidad, en el contexto de ingeniería mecánica el término de síntesis se entiende como el proceso de diseño de una máquina o sistema mecánico [2]. Existen diferentes tipos de síntesis, en el presente trabajo se abordará la síntesis dimensional.

La síntesis dimensional de un mecanismo *es la determinación de los tamaños (longitudes) de los eslabones necesarios para realizar los movimientos deseados*[1]. En este sentido, es importante determinar la tarea a realizar por el mecanismo para determinar que tipo de síntesis dimensional se llevará a cabo:

generación de función, trayectoria o movimiento. En la síntesis para generación de función, se realiza la correlación de un movimiento de entrada con un movimiento de salida en un mecanismo; en lo que respecta a la generación de trayectoria se define como el control de un punto en el plano tal que siga alguna trayectoria prescrita y finalmente, la generación de movimiento se define como el control de una línea en el plano, tal que asume algún conjunto prescrito de posiciones secuenciales [1].

La forma matemática para expresar un problema de optimización es:

$$minimizar/maximizar \quad f(\boldsymbol{x}) \tag{1}$$

sujeto a las restricciones:

$$g_j(\boldsymbol{x}) \leq 0, \qquad j = 1, 2, ..., p \tag{2}$$

$$h_j(\boldsymbol{x}) = 0, \qquad j = 1, 2, ..., q \tag{3}$$

donde $\boldsymbol{x}$ es el vector de variables de dimensión n, $f(\boldsymbol{x})$ es la función objetivo, $g_j(\boldsymbol{x})$ es el conjunto de p restricciones de desigualdad y finalmente $h_j(\boldsymbol{x})$ es el conjunto de q restricciones de igualdad.

El diseño de un mecanismo de cuatro barras que siga una trayectoria rectilínea es un caso típico de optimización *dura*. Los problemas duros de optimización son aquellos que no pueden resolverse de manera óptima o hasta un límite garantizado por medio de métodos determinísticos en un tiempo aceptable. Las metaheurísticas son algoritmos diseñados para resolver de manera aproximada un rango amplio de problemas duros de optimización. En general, las metaheurísticas tienen las siguientes características: están inspiradas en la naturaleza, hacen uso de componentes estocásticos (involucrando variables aleatorias), y tienen una serie de parámetros que requieren ser ajustados al problema a resolver [3].

Para la síntesis de mecanismos seguidores de cuatro barras se han utilizado diversas metaheurísticas en forma de algoritmos evolutivos. En [4], Cabrera et al. aplican el Algoritmo Genético (GA) con modificaciones para el manejo de restricciones, mientras que Bulatović y Dordević [5] emplean Evolución Diferencial (DE) y un método de control variable de desviaciones. En [6], Acharyya y Mandal presentan un comparativo de resultados entre tres algoritmos evolutivos diferentes para la solución del problema(Algoritmo Genético, Evolución Diferencial y Optimización por Cúmulo de Partículas (PSO)), en tanto que Matekar et al. [7] utilizan Evolución Diferencial y una función modificada de error.

En el presente trabajo se utiliza el algoritmo de Evolución Diferencial para la síntesis óptima de un mecanismo de cuatro barras seguidor de trayectoria, el cual es una versión modificada que incluye las reglas de factibilidad de Deb [8] en la etapa de competencia entre la población.La DE es un método usado ampliamente en problemas de optimización global, debido a su rápida convergencia y facilidad de implementación [9]. Por su parte, los criterios de Deb permiten mejorar la selección entre las generaciones, eligiendo al individuo más factible,

y sustituyendo así al método convencional de escoger al individuo con el mejor valor de la función objetivo.

Este artículo está organizado de la siguiente forma: la Sección 2 describe el problema de síntesis de mecanismos, con una breve explicación de la cinemática tanto del mecanismo como de su acoplador. En la Sección 3 se presentan las estrategias de optimización, analizando a la función objetivo y a las restricciones de diseño. Posteriormente, en la Sección 4 se trata el diseño óptimo del mecanismo, detallando las variables de diseño e introduciendo el problema de optimización. En la Sección 5 se muestra el algoritmo empleado, enfatizando las partes modificadas con respecto a la evolución diferencial tradicional y algunos aspectos de la implementación computacional. Finalmente, en la Sección 6 se presentan y discuten los resultados obtenidos, y en la Sección 7 se exponen las conclusiones correspondientes.

2. Problema de síntesis del mecanismo

Sea el mecanismo de cuatro barras que se muestra en la Figura 1, integrado de los siguientes elementos: barra de referencia (r_1), barra de entrada o manivela (r_2), biela o acoplador (r_3) y barra de salida o balancín (r_4).

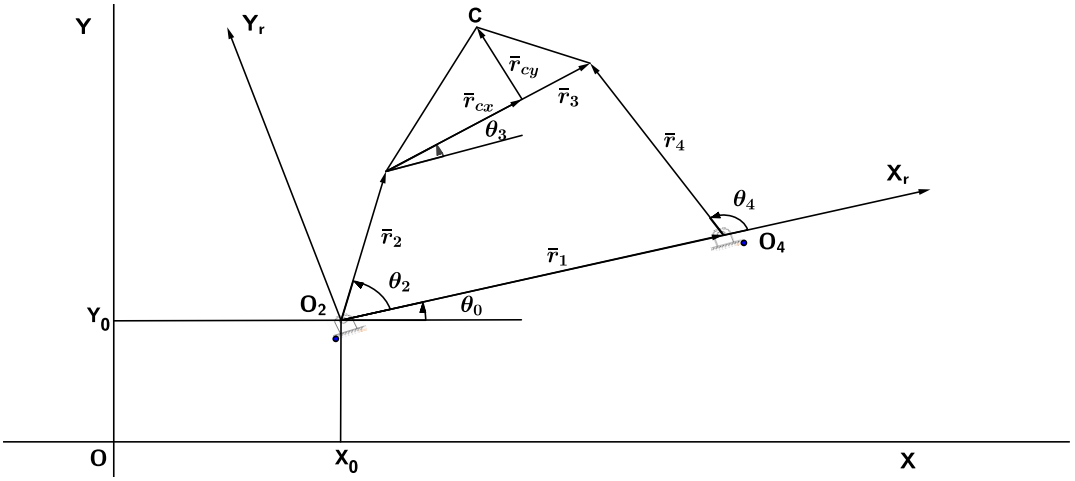


Fig. 1. Mecanismo de cuatro barras.

Con el propósito de establecer la cinemática del mecanismo se proponen dos sistemas coordenados: el fijo al mundo real denominado $0XY$ y el de referencia denominado $0X_rY_r$; donde (x_0, y_0) es la distancia entre los orígenes de ambos sistemas coordenados, θ_0 es el ángulo de rotación del sistema de referencia y θ_i en el que $i = 1, 2, 3, 4$ corresponde a los ángulos de las barras del mecanismo;

finalmente, el punto C del acoplador se determina mediante las coordenadas (r_{cx}, r_{cy}).

El problema que se resuelve en el presente trabajo es el de la generación de trayectoria para un conjunto de posiciones o puntos de precisión del acoplador sin sincronización prescrita. Esto es, el punto C del acoplador debe *tocar* un número N de puntos en forma consecutiva sin una secuencia establecida en la barra de entrada para alcanzar dichas posiciones.

2.1. Cinemática del mecanismo

La cinemática del mecanismo de cuatro barras ha sido extensamente estudiada, una explicación detallada de la misma se puede consultar en [10]. Para el presente trabajo se considera el análisis de posición del mecanismo.

Del mecanismo propuesto se puede establecer la ecuación de cierre de circuito como:

$$\boldsymbol{r}_1 + \boldsymbol{r}_4 = \boldsymbol{r}_2 + \boldsymbol{r}_3 \tag{4}$$

Aplicando notación polar a cada término de (4), se obtiene:

$$r_1 e^{j\theta_1} + r_4 e^{j\theta_4} = r_2 e^{j\theta_2} + r_3 e^{j\theta_3} \tag{5}$$

Utilizando la ecuación de Euler en (5) y separando la parte real e imaginaria:

$$\begin{aligned} r_1 cos\theta_1 + r_4 cos\theta_4 &= r_2 cos\theta_2 + r_3 cos\theta_3 \\ r_1 sin\theta_1 + r_4 sin\theta_4 &= r_2 sin\theta_2 + r_3 sin\theta_3 \end{aligned} \tag{6}$$

Para obtener la posición angular θ_3, el lado izquierdo del sistema de ecuaciones (6) se expresa en términos de θ_4:

$$\begin{aligned} r_4 cos\theta_4 &= r_2 cos\theta_2 + r_3 cos\theta_3 - r_1 cos\theta_1 \\ r_4 sin\theta_4 &= r_2 sin\theta_2 + r_3 sin\theta_3 - r_1 sin\theta_1 \end{aligned} \tag{7}$$

Elevando al cuadrado (7) y sumando sus términos, se obtiene la ecuación de Freudenstein en forma compacta [2], la cual se establece como:

$$A_1 cos\theta_3 + B_1 sin\theta_3 + C_1 = 0 \tag{8}$$

donde:

$$A_1 = 2r_3 \left(r_2 cos\theta_2 - r_1 cos\theta_1\right) \tag{9}$$

$$B_1 = 2r_3 \left(r_2 sin\theta_2 - r_1 sin\theta_1\right) \tag{10}$$

$$C_1 = r_1^2 + r_2^2 + r_3^2 - r_4^2 - 2r_1 r_2 cos\left(\theta_1 - \theta_2\right) \tag{11}$$

El ángulo θ_3 puede ser calculado como una función de los parámetros A_1, B_1, C_1 y θ_2. Dicha solución puede ser obtenida al expresar $sin\theta_3$ y $cos\theta_3$ en términos de $tan\left(\frac{\theta_3}{2}\right)$ como sigue:

$$sin\theta_3 = \frac{2tan(\frac{\theta_3}{2})}{1+tan^2(\frac{\theta_3}{2})}\ ,\ cos\theta_3 = \frac{1-tan^2(\frac{\theta_3}{2})}{1+tan^2(\frac{\theta_3}{2})} \tag{12}$$

sustituyendo éstas en (8), se obtiene una ecuación lineal de segundo orden:

$$[C_1 - A_1]\, tan^2\left(\frac{\theta_3}{2}\right) + [2B_1]\, tan\left(\frac{\theta_3}{2}\right) + A_1 + C_1 = 0 \tag{13}$$

Resolviendo (13), la posición angular θ_3 esta dada por (14).

$$\theta_3 = 2arctan\left[\frac{-B_1 \pm \sqrt{B_1^2 + A_1^2 - C_1^2}}{C_1 - A_1}\right] \tag{14}$$

Un procedimiento similar al anterior se debe llevar a cabo para obtener θ_4. A partir de (6) se obtiene la ecuación de Freudenstein que esta dada en forma compacta por:

$$D_1 cos\theta_4 + E_1 sin\theta_4 + F_1 = 0 \tag{15}$$

donde:

$$D_1 = 2r_4\,(r_1 cos\theta_1 - r_2 cos\theta_2) \tag{16}$$
$$E_1 = 2r_4\,(r_1 sin\theta_1 - r_2 sin\theta_2) \tag{17}$$
$$F_1 = r_1^2 + r_2^2 + r_4^2 - r_3^2 - 2r_1 r_2 cos\,(\theta_1 - \theta_2) \tag{18}$$

Por lo tanto, la posición angualr θ_4 esta dada por (19).

$$\theta_4 = 2arctan\left[\frac{-E_1 \pm \sqrt{D_1^2 + E_1^2 - F_1^2}}{F_1 - D_1}\right] \tag{19}$$

En las ecuaciones (14) y (19) se debe elegir el signo apropiado del radical de acuerdo al tipo de configuración del mecanismo de cuatro barras. La Tabla 1 muestra los signos de los radicales de acuerdo a la configuración del mecanismo.

Tabla 1. Selección del signo del radical de acuerdo al tipo de mecanismo

Configuración del mecanismo de cuatro barras	θ_3	θ_4
abierta	$+\sqrt{}$	$-\sqrt{}$
cruzada	$-\sqrt{}$	$+\sqrt{}$

2.2. Cinemática del acoplador

El punto de interés en el acoplador del mecanismo es C, para determinar su posición se tiene que establecer en el sistema $0X_rY_r$ que:

$$\begin{aligned} C_{xr} &= r_2 cos\theta_2 + r_{cx} cos\theta_3 - r_{cy} sin\theta_3 \\ C_{yr} &= r_2 sin\theta_2 + r_{cx} sin\theta_3 + r_{cy} cos\theta_3 \end{aligned} \tag{20}$$

Desde el sistema de coordenadas global, dicho punto se expresa como sigue:

$$\begin{bmatrix} C_x \\ C_y \end{bmatrix} = \begin{bmatrix} cos\theta_0 & -sin\theta_0 \\ sin\theta_0 & cos\theta_0 \end{bmatrix} \begin{bmatrix} C_{xr} \\ C_{yr} \end{bmatrix} + \begin{bmatrix} x_0 \\ y_0 \end{bmatrix} \tag{21}$$

Es importante hacer notar que las ecuaciones (20), (21) y las ecuaciones de la cinemática del mecanismo son suficientes para obtener la posición del punto C a lo largo de toda la trayectoria del mecanismo.

3. Estrategias de optimización

Una vez que la cinemática del mecanismo se ha establecido apropiadamente, el problema de diseño se debe definir como un problema de optimización numérica, por lo que se deben especificar las relaciones matemáticas que permitan evaluar el desempeño del sistema.

3.1. Función objetivo

Como se ha mencionado previamente, en este trabajo se desea determinar el valor de las longitudes de las barras del mecanismo, el ángulo de rotación del sistema de referencia, la distancia entre sistemas de referencia y el conjunto de ángulos del eslabón de entrada que permiten alcanzar los N puntos de precisión.

En el sistema global de coordenadas $0XY$, el i-esimo punto de precisión se indica como:

$$C_d^i = \left[C_{xd}^i, C_{yd}^i\right]^T \tag{22}$$

Por otro lado, el conjunto de N puntos de precisión se define como:

$$\Omega = \left\{C_d^i | i \in N\right\} \tag{23}$$

Entonces, dado un conjunto de valores de las barras del mecanismo y sus parámetros x_0, y_0, θ_0, cada punto del acoplador se puede expresar como una función de la posición de la barra de entrada.

$$C^i = \left[C_x(\theta_2^i), C_y(\theta_2^i)\right]^T \tag{24}$$

Por lo tanto, se desea minimizar la distancia entre los puntos de precisión C_d^i y los puntos C^i que alcanza el mecanismo en la configuración calculada.

Para cuantificar dicha distancia en todos los puntos de precisión, se propone la siguiente función:

$$f(\theta_2^i) = \sum_{i=1}^{N} \left[(C_{xd}^i - C_x^i)^2 + (C_{yd}^i - C_y^i)^2 \right] \tag{25}$$

3.2. Restricciones de diseño

Uno de los aspectos más importantes en el diseño del mecanismo es cumplir con el conjunto de restricciones que se imponen a su funcionamiento, las cuales están relacionadas con criterios de movilidad y dimensiones del mismo.

Ley de Grashof. Una de las consideraciones de mayor importancia cuando se diseña un mecanismo es la ley de Grashof, que establece los criterios para los cuales un mecanismo plano de cuatro barras puede asegurar movilidad completa de al menos una de sus barras. La ley de Grashof afirma que *para un eslabonamiento plano de cuatro barras, la suma de las longitudes más corta y más larga de los eslabones no puede ser mayor que la suma de las longitudes de los dos eslabones restantes, si se desea que exista una rotación relativa continua entre dos elementos*[2]. Si denotamos s a la longitud del eslabón más corto, l a la del más largo y finalmente p y q a las longitudes de los eslabones restantes, la ley de Grashof se establece como:

$$l + s \leq p + q \tag{26}$$

En el presente trabajo, la ley de Grashof está dada por:

$$r_1 + r_2 \leq r_3 + r_4 \tag{27}$$

Adicionalmente, con el objetivo de asegurar que el método de solución produce mecanismos que cumplen con la ley de Grashof, se establecen la siguientes restricciones:

$$r_2 < r_3 \tag{28}$$

$$r_3 < r_4 \tag{29}$$

$$r_4 < r_1 \tag{30}$$

Secuencia de ángulos de entrada. Debido a que el problema de síntesis que se aborda en el presente trabajo es el de generación de trayectoria sin sincronización prescrita, es necesario asegurar que los valores de los ángulos de la manivela sean ordenados en forma ascendente o descendente. Si denotamos el valor del ángulo de la manivela en el $i - esimo$ punto de precisión como θ_2^i, entonces se debe cumplir que:

$$\theta_2^1 < \theta_2^2 < ... < \theta_2^N \tag{31}$$

donde N es el número de puntos de precisión.

4. Diseño óptimo del mecanismo

Para la obtención del conjunto óptimo de los parámetros de diseño se debe parametrizar el sistema de acuerdo a las variables involucradas en el mismo. Una descripción apropiada de variables permite al diseñador una amplia posibilidad de reconfiguración del sistema.

4.1. Variables de diseño

Sea el vector de variables de diseño para el mecanismo de cuatro barras, establecido como:

$$\boldsymbol{p} = [p_1, p_2, p_3, p_4, p_5, p_6, p_7, p_8, p_9, p_{10}, p_{11}, p_{12}, p_{13}, p_{14}, p_{15}]^T \tag{32}$$

$$= \left[r_1, r_2, r_3, r_4, r_{cx}, r_{cy}, \theta_0, x_0, y_0, \theta_2^1, \theta_2^2, \theta_2^3, \theta_2^4, \theta_2^5, \theta_2^6\right]^T \tag{33}$$

donde las primeras cuatro variables corresponden a las longitudes de la barras del mecanismo, las siguientes dos al punto C del acoplador, las tres subsecuentes a la orientacion entre sistemas coordenados de referencia y las últimas seis a la secuencia de valores del ángulo de la barra de entrada del mecanismo.

4.2. Problema de optimización

Sea el problema de optimización numérica mono objetivo descrito por (34) hasta (47), para obtener la solución al problema de diseño de síntesis para generación de trayectoria de un mecanismo de cuatro barras:

$$Min \quad f(\boldsymbol{p}) = \sum_{i=1}^{N} \left[(C_{xd}^i - C_x^i)^2 + (C_{yd}^i - C_y^i)^2\right] \tag{34}$$

$$\boldsymbol{p} \in \mathbb{R}^{15}$$

sujeto a:

$$g_1(\boldsymbol{p}) = p_1 + p_2 - p_3 - p_4 \leq 0 \tag{35}$$

$$g_2(\boldsymbol{p}) = p_2 - p_3 \leq 0 \tag{36}$$

$$g_3(\boldsymbol{p}) = p_3 - p_4 \leq 0 \tag{37}$$

$$g_4(\boldsymbol{p}) = p_4 - p_1 \leq 0 \tag{38}$$

$$g_5(\boldsymbol{p}) = p_{10} - p_{11} \leq 0 \tag{39}$$

$$g_6(\boldsymbol{p}) = p_{11} - p_{12} \leq 0 \tag{40}$$

$$g_7(\boldsymbol{p}) = p_{12} - p_{13} \leq 0 \tag{41}$$

$$g_8(\boldsymbol{p}) = p_{13} - p_{14} \leq 0 \tag{42}$$

$$g_9(\boldsymbol{p}) = p_{14} - p_{15} \leq 0 \tag{43}$$

con las cotas:

$$0 \leq p_i \leq 60, \quad i = 1, 2, 3, 4 \tag{44}$$
$$-60 \leq p_i \leq 60, \quad i = 5, 6, 8, 9 \tag{45}$$
$$0 \leq p_i \leq 2\pi, \quad i = 7, 10, 11, 12, 13, 14, 15 \tag{46}$$

y los puntos de precisión:

$$\Omega = \{(20, 20), (20, 25), (20, 30), (20, 35), (20, 40), (20, 45)\} \tag{47}$$

5. Algoritmo de optimización

El algoritmo de Evolución Diferencial (DE) es uno de los paradigmas mas populares dentro de la computación evolutiva, ya que resuelve de manera eficiente problemas no lineales, no diferenciables y multimodales [11]. DE parte de una población inicial de soluciones candidatas arbitrarias, y en cada generación se producen individuos de prueba aplicando los operadores de reproducción (cruza y mutación). La aptitud de cada individuo nuevo se evalúa para que compita con el individuo padre, y así determinar cuál de ellos se conservará para la generación siguiente. Una de las principales ventajas de la evolución diferencial es su número reducido de parámetros de control. Solamente se requieren tres parámetros de entrada para controlar el proceso de búsqueda; esto es, el tamaño de la población o conjunto de soluciones N, la constante de diferenciación F que controla la amplificación de la variación diferencial y el parámetro de control de cruza CR [3]. Las características generales de esta técnica son:

- Representación de individuos
- Selección de padres
- Recombinación o cruza
- Mutación
- Selección de sobrevivientes y variantes.

El pseudocódigo correspondiente a la evolución diferencial se muestra en el Algoritmo 5.1:

Algoritmo 5.1: Evolución Diferencial

```
Generar una población inicial aleatoria de tamaño NP;
Evaluar la aptitud de la población inicial;
repeat
    seleccionar un padre y dos individuos adicionales;
    realizar la cruza;
    generar hijo con mutación uniforme;
    evaluar la aptitud del hijo generado;
    if el hijo es mejor que el padre then
        hijo reemplaza al padre en la siguiente generación
    else
        continúa individuo origen
until satisfacer condición de paro o terminar total de generaciones;
```

En la etapa de competencia para sustitución generacional se utilizaron las reglas de factibilidad de Deb [8]:

1. Entre dos individuos factibles, se escoge al que tenga la mejor función objetivo.
2. Entre un individuo factible y otro no factible, se escoge al factible.
3. Entre dos individuos no factibles, se escoge al que tenga un valor menor en la suma de violaciones a las restricciones.

5.1. Implementación computacional

La implementación del algoritmo se programó en MATLAB R2013a, y las corridas se llevaron a cabo en una plataforma computacional con las siguientes características: procesador Intel Core i7 @ 1.75 GHz, con 8Gb de memoria RAM y sistema operativo Windows 8.

En el programa se utilizan dos funciones, *secuencia* y *error_puntero*. En *secuencia* se evalúan las restricciones en el orden de los ángulos θ_2 correspondientes a los puntos a seguir por el mecanismo; el incumplimiento de estas restricciones se convierte en un solo valor, cuya magnitud indica el grado de violación de la secuencia. Por su parte, en *error_puntero* se observa que el individuo sea factible calculando la suma de las violaciones a las restricciones derivadas de la Ley de Grashof; si el vector de variables propuesto es factible se calcula la funcion objetivo, en caso contrario se le asigna un valor muy grande, F.O=1000. Existe una situación especial en la que el individuo es factible pero produce un mecanismo de doble balancín, en cuyo caso se penaliza la funcion objetivo.

6. Resultados

Se llevó a cabo un conjunto de cincuenta corridas del algoritmo propuesto, de las cuales se seleccionaron las diez con mejores resultados de la función objetivo. Inicialmente, se realizó la calibración de los parámetros del algoritmo

buscando una mayor convergencia del resultado dentro de la zona factible. La puesta a punto se obtuvo con el siguiente conjunto de parámetros: tamaño de la población NP = 100, número máximo de generaciones GMAX = 5,000, factor de escalamiento F = [0.8, 1], y factor de cruza CR = [0.3, 0.9]; estos dos últimos valores se generaron en forma aleatoria dentro de los rangos especificados.

Las Tablas 2 y 3 muestran los valores de los vectores solución obtenidos en las mejores ejecuciones. Como puede observarse, todos los valores caen dentro de los límites marcados por las restricciones de diseño; por cuestiones de espacio sólo se muestran los primeros tres dígitos decimales, aunque en la corrida de las simulaciones se generaron datos con una precisión de catorce posiciones decimales. La Tabla 4 muestra la magnitud del error obtenido por el algoritmo, con un excelente valor promedio del orden de 10^{-14} e incluso alcanzando el nivel de 10^{-29} para la mejor corrida. Finalmente, en la Tabla 5 se pueden observar los tiempos de ejecución correspondientes, los cuales son relativamente cortos, en el orden de 2.6 min.

Tabla 2. Mejores diez vectores de solución.

N	r_1	r_2	r_3	r_4	r_{cx}	r_{cy}	θ_0	x_0
1	38.463	8.544	27.892	37.388	36.897	18.156	3.943	-9.429
2	38.045	8.517	27.934	38.038	37.631	16.453	3.952	-9.323
3	38.003	8.384	28.182	36.679	37.614	18.679	3.965	-10.314
4	30.642	7.877	28.909	30.619	47.306	19.146	4.253	-21.061
5	37.415	8.538	27.102	37.227	36.230	16.332	3.940	-8.2637
6	38.039	8.394	28.628	37.501	38.875	17.736	3.978	-10.915
7	35.483	8.052	28.653	34.413	40.594	19.211	4.056	-13.694
8	37.568	8.338	28.556	36.949	38.986	17.899	3.988	-11.191
9	32.332	7.730	27.220	31.033	39.811	19.179	4.108	-13.829
10	36.624	8.184	28.110	35.295	38.323	18.826	4.001	-11.270

Con respecto al análisis de convergencia en las corridas seleccionadas, en la Figura 2 se aprecia que el valor de la función objetivo tiende a converger aproximadamente después de las primeras cincuenta generaciones, mientras que en la Figura 3 se observa que la totalidad de las soluciones (individuos) entran a la zona factible en ese mismo lapso. Ambas gráficas se ajustan al comportamiento general de los algoritmos evolutivos [9] y demuestran la capacidad de la evolución diferencial para localizar el mínimo global en un tiempo corto.

7. Conclusiones

En este articulo se presentó una técnica para la síntesis de un mecanismo de cuatro barras para seguimiento de trayectorias, con base en el algoritmo de Evolución Diferencial. Se realizó una modificación al algoritmo original para aplicar los criterios de factibilidad de Deb en la etapa de concurso entre los

Tabla 3. Mejores vectores de solución (cont.) y función objetivo correspondiente.

y_0	${\theta_2}^1$	${\theta_2}^2$	${\theta_2}^3$	${\theta_2}^4$	${\theta_2}^5$	${\theta_2}^6$	F.O.
59.705	1.625	2.421	2.936	3.429	3.965	5.325	5.048E-29
59.223	1.750	2.465	2.976	3.472	4.021	5.126	5.048E-29
59.911	1.625	2.422	2.940	3.436	3.977	5.227	6.310E-29
59.641	1.912	2.481	2.946	3.396	3.869	4.465	7.573E-29
58.625	1.689	2.442	2.958	3.457	4.004	5.262	1.161E-27
59.960	1.741	2.459	2.969	3.464	4.009	5.048	5.452E-27
59.989	1.735	2.445	2.956	3.448	3.984	4.873	2.930E-22
59.894	1.735	2.455	2.967	3.462	4.007	5.028	8.707E-18
58.658	1.673	2.414	2.936	3.435	3.976	4.848	5.161E-17
59.697	1.646	2.424	2.945	3.445	3.991	5.092	1.774E-13

Tabla 4. Estadísticas de las simulaciones numéricas.

Mejor	5.04870E-29
Promedio	1.77509E-14
Peor	1.77449E-13
Desviación estándar	5.32328E-14

Tabla 5. Tiempo de ejecución de las simulaciones numéricas.

Simulación	1	2	3	4	5	6	7	8	9	10
Tiempo (min)	2.63	2.61	2.76	2.63	2.63	2.60	2.65	2.65	2.63	2.58

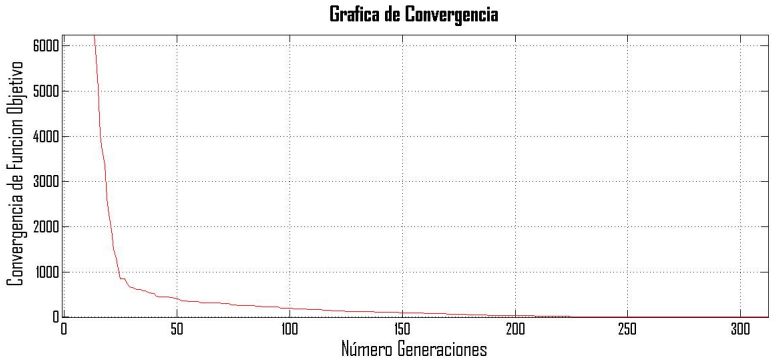


Fig. 2. Convergencia de la función objetivo hacia el valor óptimo.

individuos originales y los de nueva creación, que produjo una mejora en la toma de decisión para conformar la generación siguiente. De los resultados obtenidos, se pudo establecer que este método de solución para la optimización de mecanismos produce buenos resultados desde el punto de vista del diseño en

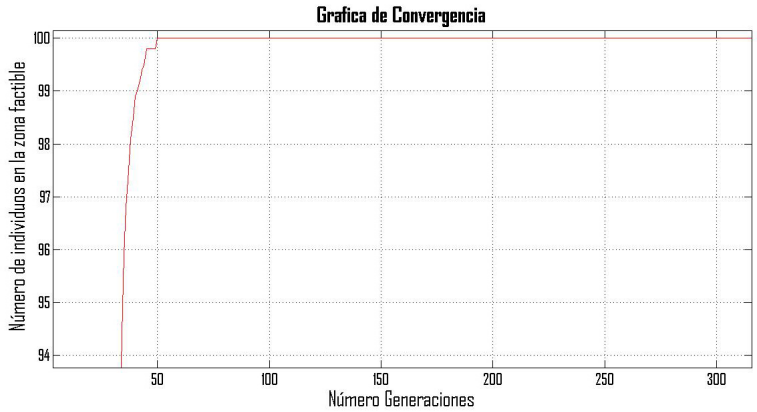


Fig. 3. Promedio de individuos factibles por generación, considerando 5000 generaciones y 100 individuos.

ingenieria, sin requerir el uso de recursos de cómputo extensivos y con un tiempo de respuesta razonable. Si bien el algoritmo aquí propuesto fue desarrollado para el caso específico de un mecanismo de cuatro barras, su implementación simple permite que pueda utilizarse para diseñar otro tipo de mecanismos. En este sentido, la dificultad mayor radica en la correcta interpretación y formulación de las restricciones del problema específico. Así mismo, la sintonización de los parámetros del algoritmo requiere atención especial; la generación aleatoria de los mismos produce buenos resultados, aunque en algunos casos es necesario un ajuste más preciso.

Agradecimientos. El primer autor es el autor para correspondencia. Todos los autores agradecen el apoyo del Instituto Politécnico Nacional a través de la SIP vía el proyecto SIP-20141257. El segundo autor agradece al CONACyT por la beca para estudios de posgrado en el CIDETEC-IPN.

Referencias

1. Norton, R.L.: Diseño de Maquinaria, una Introducción a la Síntesis y al Análisis de Mecanismos y Máquinas, McGraw Hill, México (1995)
2. Shigley, J.E., Uicker, J.J. Jr.: Teoría de Máquinas y Mecanismos, McGraw Hill, México (1988)
3. Boussaid, I., Lepagnot, J., Siarry, P.: A Survey on Optimization Metaheuristics. Information Sciences. 237, 82–117 (2013)
4. Cabrera, J.A., Simon, A., Prado, A.: Optimal Synthesis of Mechanisms with Genetic Algorithms. Mechanism and Machine Theory. 37, 1165–1177 (2002)
5. Bulatović, R.R., Dordević, S.R.: On the Optimum Synthesis of a Four-Bar Linkage Using Differential Evolution and Method of Variable Controlled Deviations. Mechanism and Machine Theory. 44, 235–246 (2009)

6. Acharyya, S.K., Mandal, M.: Performance of EAs for Four-Bar Linkage Synthesis. Mechanism and Machine Theory. 44, 1784–1794 (2009)
7. Matekar, S.B., Gogate, G.R.,: Optimum Synthesis of Path Generating Four-Bar Mechanisms Using Differential Evolution and a Modified Error Function. Mechanism and Machine Theory. 52, 158–179 (2012)
8. Deb, K.: An Efficient Constraint Handling Method for Genetic Algorithms. Computer Methods in Applied Mechanics and Engineering. 186, 311–338 (2000)
9. Eiben, A.E., Smith, J.E.: Introduction to Evolutionary Computing, Springer Verlag, New York (2007)
10. Pérez Moreno, R.: Análisis de Mecanismos y Problemas Resueltos, Alfaomega Grupo Editor S.A. de C.V., México (2006)
11. Price, K.V.: An Introduction to Differential Evolution, New Ideas in Optimization, Mc Graw Hill, UK (1999)

Algoritmo de optimización por cúmulo de partículas para la construcción de redes de ordenamiento rápidas

Blanca López, Nareli Cruz-Cortés

Centro de Investigación en Computación,
Instituto Politécnico Nacional (CIC-IPN), DF, México

blopezb10@sagitario.cic.ipn.mx, nareli@cic.ipn.mx

Resumen. Una Red de Ordenamiento (RO) es un método de ordenamiento *oblivious* que permite ordenar N objetos mediante el uso de un número fijo de comparadores binarios los cuales están distribuidos en un orden predefinido. Por décadas, el diseño de las RO ha capturado la atención de muchos investigadores quienes han dedicado un gran esfuerzo por minimizar el número de comparadores para una entrada dada N. En este trabajo, proponemos una estrategia novedosa y sencilla para diseñar RO que maximicen el número de comparadores en paralelo, lo que es equivalente a encontrar RO rápidas. La heurística bioinspirada llamada Optimización por Cúmulo de partículas (PSO) es empleada para guiar el proceso de búsqueda. Nuestros resultados muestran que el esquema propuesto permite construir RO rápidas y en algunos casos se encuentran RO óptimas por el número de comparadores. Los diseños de las RO construidas cuentan con diferentes configuraciones a las RO rápidas conocidas en la literatura especializada.

Palabras clave: Redes de ordenamiento, optimización por cúmulo de partículas, ordenamiento.

1. Introducción

Diseñar Redes de Ordenamiento (RO) es un problema combinatorio que ha sido estudiado durante décadas [17,8]. Las RO son un ejemplo de algoritmos llamados *oblivious*, esto es, que los pasos necesarios para ordenar no dependen de los anteriores. La operación fundamental en una RO es la llamada comparación-intercambio y se conoce en la literatura especializada como *comparador*. Un conjunto de comparadores conforman una RO. Un comparador C recibe dos datos de entrada (a, b) para ser comparados, si $a > b$ entonces se aplica la operación de intercambio donde $a < b$. Una RO puede ser interpretada como un circuito para ordenar una secuencia de datos [16].

Generalmente las RO son representadas esquemáticamente como en la Figura 1. Cada línea horizontal representa el trayecto de un dato de entrada.

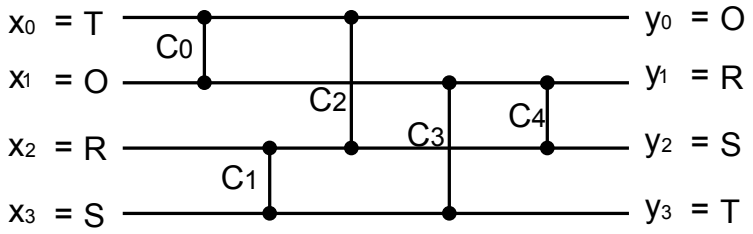


Fig. 1. Ejemplo de una Red de Ordenamiento para 4 datos de entrada.

Cada línea vertical es un comparador que evalúa los elementos entre su extremo superior e inferior. Si el valor del extremo superior es más grande que el extremo inferior entonces intercambia los datos. Por ejemplo, en una RO donde se quieren ordenar 4 datos de entrada, los valores de x_0, x_1, x_2, x_3 son colocados del lado izquierdo de la RO y un dato por cada línea horizontal, como en la Figura 1. Las líneas verticales etiquetadas como C_0, C_1, C_2, C_3, y C_4 son los comparadores. Cada comparador recibe dos valores, esto es, C_0 recibe los valores x_0 y x_1; C_1 recibe los valores de x_2 y x_3, y así sucesivamente. De esta forma, los comparadores son ejecutados de izquierda a derecha según su aparición. En este ejemplo primero se ejecuta C_0, posteriormente C_1, en seguida C_2, entonces C_3 y finalmente C_4. Esto es, C_0 evalúa 'T'>'R' entonces, los valores de x_0 y x_1 son intercambiados. El siguiente comparador es C_1 que evalúa 'O'>'S' mantiene a x_0 y x_1 sin intercambio. En seguida C_3 evalúa 'R'>'O' entonces, los valores x_0 y x_2 son intercambiados. Nótese que es posible ejecutar los comparadores C_0 y C_1 al mismo tiempo. De manera análoga los comparadores C_2 y C_3 podrían aplicarse en forma paralela. Este proceso continúa hasta que todos los comparadores son aplicados. Finalmente, la lista ordenada y_0, y_1, y_2, y_3 se obtiene al lado derecho de la RO y debe cumplir con $y_0 \leq y_1 \leq y_2 \leq y_3$.

La independencia de los datos de entrada se debe a la configuración interna de las RO. Es por este motivo que las RO se encuentran en la clasificación de algoritmos no adaptativos. Las principales diferencias entre las RO y los algoritmos clásicos de ordenamiento (tales como el *bubblesort*, *sellsort*, *quicksort*) son las siguientes:

1. La RO cuenta con una cantidad fija de comparadores necesaria para ordenar cualquier permutación de datos de entrada;
2. El número de pasos necesarios para ordenar es fijo *a priori*.

En realidad, el diseño de una RO de tamaño mínimo incrementa considerablemente su complejidad cuando el número de entradas es incrementado. Algunas RO que han sido ampliamente estudiadas son $n = 10$, $n = 12$, y $n = 16$. Además en la literatura las publicaciones son orientadas en su mayoría a la construcción de RO minimizando el número de comparadores. Por lo que existe poco trabajo dedicado al diseño de RO rápidas, esto es, RO con máximo paralelismo.

Diferentes trabajos se han encontrado para construir Redes de Ordenamiento rápidas, como en [16,15] donde emplean estrategias empleadas en las técnicas

clásicas de ordenamiento, tal como el *sellsort* y el *mergesort* paralelo respectivamente. Además en [1] fue propuesta una estrategia recursiva. Por otro lado, el trabajo publicado por Batcher en [11,3] es una estrategia eficiente llamada $Odd - Even\ sort$, la cual ayuda a construir RO para $n > 16$ con algunos resultados favorables con respecto a RO rápidas.

En este trabajo, proponemos una técnica para encontrar RO que maximicen el número de comparadores por capa. Para esto proponemos una novedosa y compacta manera de representar la RO, esto ocasiona la construcción de RO rápidas convenientemente. Por otro lado, la tarea de búsqueda la lleva a cabo una técnica bioinspirada llamada algoritmo de Optimización por Cúmulo de Partículas (PSO por sus siglas en inglés), es importante señalar que no se ha estudiado la construcción de RO con PSO. Los resultados presentados muestran que la estrategia es eficiente y capaz de encontrar RO rápidas. Además, la cantidad de comparadores encontrados en las RO resultantes son competitivas con el estado del arte.

2. Algoritmo de optimización por cúmulo de partículas (PSO)

El PSO [10] es una meta-heurística bioinspirada inspirada en el comportamiento social del vuelo de las aves o el movimiento de los bancos de peces. Ésta se ha convertido en un técnica popular para resolver problemas de optimización complejos. El PSO se describe generalmente como una población de vectores (llamadas partículas) cuyas trayectorias oscilan en torno a una región. En el caso más general, teniendo en cuenta la configuración del enjambre (población), hay dos versiones del PSO: globales y locales. En la versión global (*gbest*) la trayectoria de cada partícula está influenciada por el mejor punto encontrado por cualquier miembro de todo el enjambre. En la versión local (*lbest*) la trayectoria de cada partícula puede ser influenciada sólo por el mejor miembro en el vecindario (el mismo enjambre). Además, cada partícula también se ve influida por su mejor éxito previo.

En su forma más general el PSO binario se describe como: Se denota a Np como el número de partículas en la población. Así que $Y_i^t = (y_{i1}^t, y_{i2}^t, \dots, y_{iD}^t)$, $y_{id}^t \in 0, 1$, es una partícula i con D *bits* en la iteración t, donde Y_i es una solución potencial que tiene una tasa de cambio llamada velocidad.

La velocidad es $V_i^t = (v_{i1}^t, v_{i2}^t, ..., v_{iD}^t)$, $v_{id}^t \in R$. Así que $P_i^t = (p_{i1}^t, p_{i2}^t, ..., p_{iD}^t)$ representa la mejor solución que la partícula i ha obtenido hasta la iteración t, y $p_g^t = (p_{g1}^t, p_{g2}^t, ..., p_{gD}^t)$ representa la mejor solución obtenida a partir de P_i^t en la población (*gbest*)o del vecindario local (*lbest*), en la iteración t.

Cada partícula ajusta su velocidad de acuerdo a la siguiente ecuación:

$$v_{id}^t = v_{id}^{t-1} + co_1 r_1 (p_{id}^t - y_{id}^t) + co_2 r_2 (p_{gd}^t - y_{id}^t) \tag{1}$$

donde co_1 es el factor de aprendizaje cognitivo, co_2 es el factor de aprendizaje social, y r_1 y r_2 son números generados en forma aleatoria con distribución

uniforme en $[0, 1]$. Los valores co y r determinan los pesos de dos partes donde la suma de ellos es limitada normalmente a 4 [10].

La velocidad representa la probabilidad de que el valor actual tome el valor de uno. Para mantener el valor en el intervalo $[0, 1]$ se emplea la función sigmoidal:

$$sig(v_{id}^t) = \frac{1}{1 + exp(-v_{id}^t)} \tag{2}$$

Algoritmo de PSO

Algoritmo 1 Red de ordenamiento para $n = 4$

Input: x_0, x_1, x_2, x_3

```
1    begin
2        FOR i = 0 hasta Np
3            STATE Calcular la aptitud por cada p_i
4            IF Y_i^t > P_i^t
5                FOR d = 1 hasta D bits
6                    STATE P_id^t = Y_i^t
7                END FOR
8            END IF
9            g = i
10           FOR j=índices de los vecinos
11               IF P_j^t > P_g^t
12                   STATE g = j
13               END IF
14           END FOR
15           FOR d = 1 hasta D
16               STATE v_id^t = v_id^(t-1) + co1 r1 (P_id^t - Y_id^t) + co2 r2 (P_gd^t - Y_id^t)
17               STATE v_id^t ∈ [-V_max, +V_max]
18               IF Número aleatorio [0, 1] < sig(v_id^t)
19               STATE Y_i^(t+1) = 1
20           ELSE
21               STATE Y_i^t = 0
22           END IF
23           END FOR
24       END FOR
```

Output: y_1, y_2, y_3, y_4

3. Redes de Ordenamiento: Conceptos básicos

Las RO son algoritmos que ordenan una lista de datos. Están compuestas por un grupo de comparadores, donde cada comparador ejecuta una acción

comparación-intercambio entre dos elementos (a, b). El elemento a debe ser más pequeño que el elemento b, de lo contrario será intercambiado (b, a). Los datos de salida de las RO están en una lista con un orden monotónicamente creciente.

Durante el proceso de construcción de una RO es necesario determinar si la RO candidata es válida o no. Es posible saberlo si todas las permutaciones de los datos de entrada son correctamente ordenados por la RO, de ser así, es considerada como una RO válida. Por lo tanto, para un tamaño de dato de entrada n es necesario probar $n!$ veces la RO, esto implica que la tarea de la verificación incrementa su complejidad computacional cuando n se incrementa, inclusive puede llegar a ser intratable [13]. Una alternativa es aplicar el principio cero-uno ($TheoremZ$) presentado en el libro de Knuth [11]. Este principio afirma que una RO que ordene todas las 2^n secuencias de ceros y unos en un orden decreciente, ordenará cualquier secuencia arbitraria de n números. Por lo que para $n > 2$, $n! > 2^n$, la aplicación de este principio permite contar con una forma razonable de verificación en las RO.

En general, la eficiencia de una RO es medida con base en dos criterios:

- El número de comparadores implicados; y
- La velocidad máxima que la RO pueda alcanzar, la cual es inversamente proporcional al número de capas.

Desde hace más de 5 décadas, varios investigadores se han dedicado a buscar RO con el mínimo número de comparadores. El hecho de verificar que una secuencia de comparadores es una RO válida, es un problema NP-Completo [1,14,6] y diseñar RO con un mínimo número de comparadores o maximizar el número de capas es un problema de tipo NP-hard [14]. Una ventaja de las RO es que sus comparadores pueden ser ejecutados en paralelo. Usualmente, el tiempo discreto requerido para el ordenamiento se refiere al número de pasos o capas que la RO debería considerar después de verificar la independencia de los comparadores que pueden ser ejecutados en paralelo.

Una RO es *rápida* si tiene un bajo número de capas, donde cada capa está compuesta por un subconjunto de comparadores que puedan realizar sus operaciones concurrentemente sin interferir entre ellas. Por ejemplo, en la Figura 2 se tiene el diagrama de flujo de la RO para $n = 4$ de la Figura 1. Los comparadores C_0 y C_1 pueden ser aplicados al mismo tiempo (independientes), esto permite realizar las operaciones de forma concurrente. Al igual que C_2 y C_3 podrían ejecutarse en paralelo.

El conjunto de comparadores que se ejecutan concurrentemente se les conoce como capas, denotado por L. Por ejemplo, en la RO de la Figura 3, el rectángulo a la izquierda (línea de puntos separados) representa una capa que es ejecutada en el tiempo $t1$. De tal manera que, la RO puede ser ejecutada en tres pasos. Note que una capa no puede contener más de $\lfloor n/2 \rfloor$ comparadores.

Entre las diferentes formas de construcción, las RO rápidas entre $3 \leq n \leq 16$ con su respectivo número de capas se puede ver en la Tabla 1. La segunda fila presenta el número de capas de las RO rápidas conocidas. En la tercera fila se tienen los límites mínimos de capas que han sido obtenidos de información teórica [14,17].

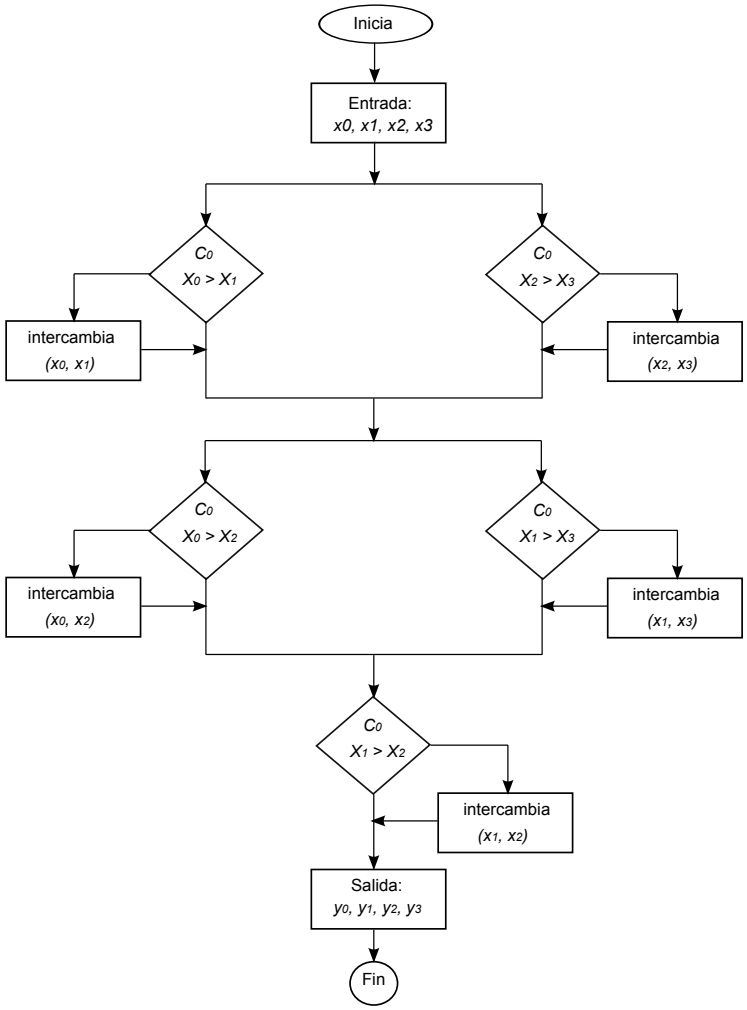


Fig. 2. Diagrama de flujo de la Red de Ordenamiento para 4 datos de entrada.

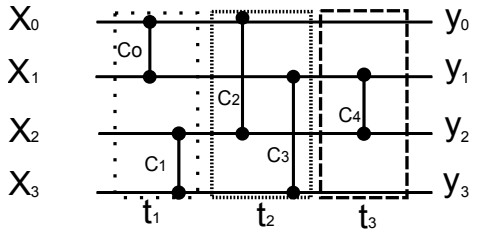


Fig. 3. Ejemplo de una RO para $n = 4$ con 3 capas.

Por otro lado el número de comparadores que tienen las RO rápidas conocidas se encuentran en la Tabla 2.

Tabla 1. RO rápidas conocidas y límites mínimos obtenidos de forma teórica.

n	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Número de capas	3	3	5	5	6	6	7	7	8	8	9	9	9	9
Límite mínimo	3	3	5	5	6	6	7	7	7	7	7	7	7	7

Tabla 2. Número de comparadores de las RO rápidas conocidas desde $n = 3$ hasta $n = 16$.

n	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Número de capas	3	3	5	5	6	6	7	7	8	8	10	10	10	9
Número de comparadores	3	5	9	12	31	19	25	31	35	40	45	53	56	61

4. Trabajo previo en Redes de Ordenamiento

Uno de las primeras investigaciones para diseñar RO fue presentado por P. N. Armstrong, R. J. Nelson y D. G. O'Connor en 1945 [2]. Allí, los autores diseñaron RO para $n = 5$, $n = 6$, $n = 7$ y $n = 8$ con 5, 9, 12 y 19 comparadores respectivamente. Posteriormente, Knuth [11] demostró que éstas son las RO más eficientes posibles (con el menor número de comparadores).

En los años 60's los investigadores trabajaron intensamente sobre la RO para $n = 16$ datos de entrada. Los resultados más relevantes fueron los que a continuación se mencionan: R. C. Bose y Nelson en 1962 [4] encontraron una RO con 65 comparadores. En 1964 K. E. Batcher y Knuth [3,16] presentaron una RO con 63 comparadores, dos años más tarde G. Shapiro obtuvo una RO con 62 comparadores. En el mismo año (1969), M. W. Green logró el diseño de una RO con sólo 60 comparadores, que sigue siendo hasta el momento la RO más eficiente para $n = 16$. Un logro en la misma década fue para la RO de $n = 9$ con 25 comparadores diseñada por Floyd en 1964 [7]. Mientras Waksman en 1969 [18] daba a conocer que hasta el momento es la RO más eficiente conocida y estudiaba la simetría que presentaban los comparadores, con este estudio diseñó una RO de $n = 10$ con 29 comparadores.

Los algoritmos presentados por Batcher en [3] fueron empleados por Shapiro y Green para trabajar con las RO para $n = 12$ donde se encontraron 39 comparadores.

Desde los años 90's, surgen nuevas propuestas. En 1995 Hugues Juillé [9], utilizó una estrategia co-evolutiva y empleó un gran número de procesadores

para encontrar la RO para $n = 13$ con sólo 45 comparadores, que es la mejor RO conocida.

Recientes estrategias orientadas a reducir la complejidad del algoritmo y el tiempo de procesamiento se han publicado desde entonces, sin embargo no han mejorado el número de comparadores en los resultados encontrados.

Las trabajos más relevantes se presentan a continuación. En 1999 J. Koza [12] diseñó la RO para $n = 7$ con Programación Genética. En 2005, Choi y Moon [5] publicaron un Algoritmo Genético para diseñar una RO para tamaños de entrada $n = 10$, $n = 12$, y $n = 16$. Los autores argumentaron que su trabajo disminuye el tiempo de procesamiento de una manera impactante para encontrar las RO mejores conocidas en segundos con un simple procesador. Sin embargo, su técnica disminuye drásticamente el espacio de búsqueda debido a que copia la mitad de los comparadores de una RO óptima (RO de Green). Por ejemplo, para $n = 12$, su técnica añade los primeros 18 comparadores, mientras que para $n = 16$, toma 32 comparadores con los que inicializa su algoritmo.

5. Propuesta

5.1. Planteamiento del problema

El problema que aborda este trabajo es el diseño de RO rápidas para tamaño de los datos de entrada de $n = 10$, $n = 12$, y $n = 14$.

5.2. Metodología

La idea general es encontrar RO con un mínimo número de capas. El tamaño de capas estará limitado por el tamaño de capas que tiene una RO válida, entonces nuestra técnica procede a llenar cada una de estas capas con una *adecuada* selección de los comparadores. En este caso una *adecuada* selección de los comparadores significa que éstos garantizarán una RO válida.

Se establece que para una entrada de datos de tamaño n, la siguiente información es conocida:

- El máximo número de comparadores en una capa es $\lfloor n/2 \rfloor$,
- El número mínimo de capas en la RO rápida (de acuerdo a la Tabla 1).

Por lo tanto, en este escenario, el problema es determinar qué comparadores debería poblar en cada una de las capas. Debido a la gran cantidad de posibilidades, se decidió utilizar una heurística para encontrar tales RO. Para este caso se presenta la versión binaria del algoritmo PSO que se presentó en la Sección 2.

La representación de las partículas, la función de aptitud, y la descripción general del algoritmo se presentan a continuación.

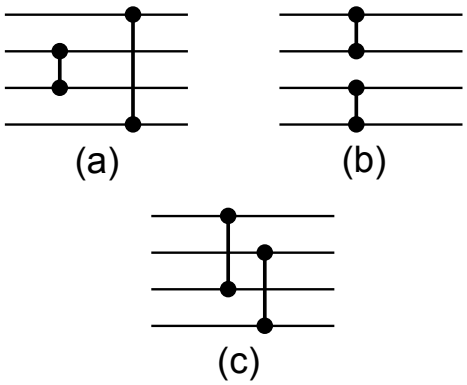


Fig. 4. Total de posibles configuraciones de una capa para una RO de $n = 4$. En (a) se muestra la configuración $\{1, 2\}\{0, 3\}$, en (b) la configuración $\{0, 1\}\{2, 3\}$, y en (c) la configuración $\{0, 2\}\{1, 3\}$.

5.3. Representación de la partícula

Una partícula representa una RO candidata compuesta por l capas (donde l es la más rápida). Cada capa se considera como una variable de decisión.

Recordemos que para que una determinada RO con entrada n, se conoce su número mínimo de capas (Tabla 1), por lo que es, precisamente, el número de capas representado en una partícula.

Para seleccionar los comparadores que poblarán una capa, todas las posibles configuraciones de capas son generadas y enumeradas, y sus números (utilizando codificación binaria) funcionará como identificadores.

Por ejemplo, supongamos que estamos buscando la RO para $n = 4$. El mínimo número de capas conocido es 3 (es el más rápido, ver Tabla 1 tercera fila, tercera columna). Por lo tanto, una partícula P se compone de 3 variables de decisión que es equivalente a 3 capas: $P = (L1, L2, L3)$

Ahora, dentro de cada capa L podemos colocar 2 comparadores paralelos (con base en que $n/2 = 2$). Es evidente que sólo hay tres posibles maneras de poner 2 comparadores paralelos, que son los siguientes (se ilustra en la Figura 4):

- Configuración 1: $\{1, 2\}\{0, 3\}$
- Configuración 2: $\{0, 1\}\{2, 3\}$
- Configuración 3: $\{0, 2\}\{1, 3\}$

La primera configuración es etiquetada con el número 1, la segunda con el 2 y la tercera con el 3. Por lo tanto, las variables de decisión pueden tomar uno de estos tres valores. Continuando con el ejemplo, la partícula

$$P = (1, 3, 1)$$

representaría a la RO

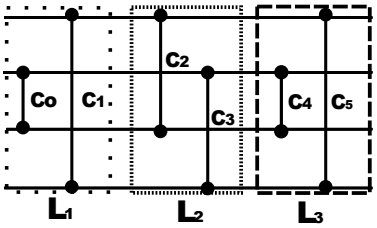


Fig. 5. Ejemplo de la representación de una partícula para $n = 4$

$$L1 = (\{1,2\}\{0,3\}), (L2 = \{0,2\}\{1,3\}), (L3 = \{1,2\}\{0,3\})$$

que se puede ver en la Figura 5. En este trabajo los valores son codificados con representación binaria.

5.4. Función de aptitud

La RO representada por una partícula *debe* ordenar todas las permutaciones de datos de entrada $x = (x_0, x_1, ..., x_{n-1})$ para reconocer su validez.

Esto es equivalente a ordenar todas las 2^n secuencias binarias (Principio cero-uno mencionado en la Sección 3). De acuerdo con este principio, se quiere minimizar el número de secuencias binarias desordenadas que deja una partícula. Así que, el mínimo valor esperado es cero.

La *aptitud* de una partícula es el número de secuencias binarias que no es capaz de ordenar. Las secuencias binarias de entrada son representadas por B_n y el conjunto de secuencias binarias ordenadas, lo llamaremos S_n.

Retomando el ejemplo con la RO para $n = 4$, los datos de entrada son $B_4 = 0000, 0001, 0010...1111$ y, S_4 es el conjunto de secuencias binarias ordenadas, así que $S_4 = 0000, 0001, 0011, 0111, 1111$. El objetivo es minimizar el conjunto de B_4 hasta que todas las secuencias estén ordenadas.

La partícula presentada en la Figura 5 tiene un valor de aptitud igual a 2 porque existen dos secuencias binarias $0010, 1011$ en B_4, que la RO no puede ordenar.

La única manera de evaluar sí una RO ordena un conjunto de secuencias, es aplicando todos sus comparadores a cada secuencia de B_n y supervisando que la salida sea correctamente ordenada.

El PSO presentado en la Sección 2 es utilizado para encontrar los comparadores que conforman cada capa L y minimizar el tamaño de B_n.

6. Experimentos y Resultados

Nuestros experimentos fueron dirigidos a diseñar RO rápidas para $n = 10$, $n = 12$ y $n = 14$ mediante la ayuda del PSO. Fueron aplicadas las dos versiones: local *lbest* y global, para cada n. La versión local del PSO (*lbest*), los vecindarios

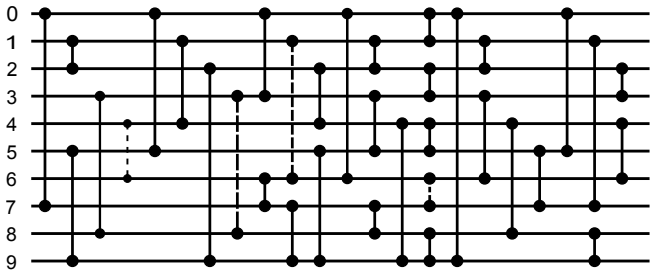


Fig. 6. Nuevo diseño para $n = 10$ encontrada por el algoritmo con el mínimo número de capas conocido igual a 7.

fueron ajustados por 10 partículas seleccionadas aleatoriamente con una configuración estática.

Todos los experimentos fueron ejecutados en un procesador Core i5-430M (2,26 GHz). Para cada caso se realizaron 20 ejecuciones independientes.

Para el diseño de $n = 10$, se llevaron a cabo experimentos con diferente número de partículas. Se encontró una RO con el PSO versión local. Ésta tiene 7 capas y 31 comparadores, resultado que coincide con la RO rápida y óptima en el número de comparadores. Además, esta RO cuenta con una configuración diferente a las conocidas (nuevo diseño). Su esquema es presentado en la Figura 6.

Nuestra técnica diseña una RO con 35 comparadores en 7 capas, de las cuales 4 comparadores son redundantes, estos comparadores no aplican la operación de intercambio durante el desempeño de la RO. Por lo tanto se consideran inútiles y son ignorados.

Es importante señalar que la técnica no recibió comparadores iniciales de ayuda, esto para disminuir el número de secuencias binarias, como se hizo en [5]. Los parámetros para el experimento fueron:

- Número de partículas 100
- Iteraciones 1000
- $V_{max} = 4.0$

Para el caso de $n = 12$, los resultados encontrados se dieron con los siguientes parámetros:

- Local PSO.
- Número de partículas 100
- Iteraciones 1000 hasta 3000
- $V_{max} = 4.0$

La técnica encuentra una RO con 51 comparadores y 9 capas. El esquema de esta RO se encuentra en la Figura 7. Inicialmente con el total de comparadores 51, aunque 7 comparadores son redundantes.

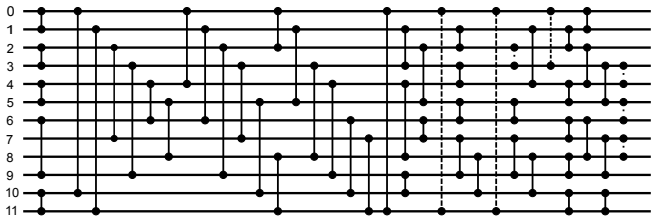


Fig. 7. La mejor RO rápida encontrada por el PSO con 51 comparadores y 9 capas. Las líneas punteadas son los comparadores redundantes.

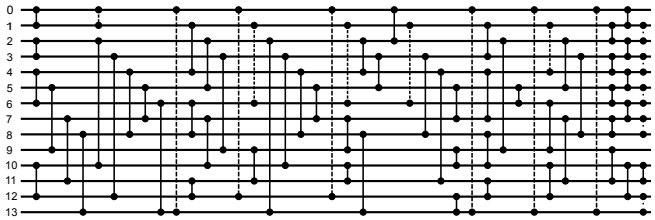


Fig. 8. La mejor RO para $n = 14$ encontrada por el PSO con 74 comparadores. Las líneas punteadas son comparadores redundantes (12). El resultado de la RO es 11 capas y 62 comparadores

.

Las capas inicialmente se fijaron con 8 pasos y al no encontrar RO válidas después de las $300,000$ evaluaciones se agregó una capa más. En la Figura 7 son identificados los comparadores redundantes con lineas punteadas. [1]

Con respecto a los experimentos para la RO con $n = 14$, la heurística encontró una RO con 11 capas y 62 comparadores. Esta solución se encuentra en la Figura 8. Esta RO es un nuevo diseño. La técnica presenta 74 comparadores, de los cuales 12 son redundantes. Estos comparadores se distinguen como líneas de punteadas.

7. Conclusiones

En general, existen dos formas como medida de eficiencia de una RO. La primera es el número de comparadores, y la segunda es el número de pasos o capas para identificar qué tan rápida es la RO, éstos se refiere a la máxima paralelización de comparadores en la RO. Considerando este último punto, se presenta una estrategia para diseñar RO de máxima paralelización de comparadores. La idea principal es fijar el tamaño de capas igual a la mínima conocida. De esta forma, los comparadores por cada capa son buscados y agregados a la

[1] Esto se debe a que la estrategia propuesta inserta el número máximo de comparadores por capa, lo que ocasiona que en algunos casos los comparadores lleguen a ser redundantes.

partícula, la cual usa una representación compacta. La técnica de búsqueda es adaptada al PSO. La RO para $n = 10$, $n = 12$ y $n = 14$ con ayuda de la heurística se encontraron nuevos diseños, unos resultados fueron muy cercanos a las RO rápidas y para la $n = 10$ además de una nueva configuración alcanza los mínimos conocidos en ambos objetivos.

Como trabajo a futuro es necesario diseñar una estrategia local para definir los resultados, especialmente durante el final del proceso. Además se puede cambiar la heurística para diseñar una estrategia local.

Agradecimientos. Las autoras agradecen el apoyo parcial de CONACYT a través del proyecto 132073. Además a la Secretaría de Investigación y Posgrado del IPN por el apoyo mediante el proyecto 20140147. De igual forma agradecemos al Instituto Politécnico Nacional.

Referencias

1. Ajtai, M., Komlós, J., Szemerédi, E.: An 0(n log n) sorting network. In: Proceedings of the fifteenth annual ACM symposium on Theory of computing. pp. 1–9. STOC '83, ACM, New York, NY, USA (1983)
2. Armstrong, P.: Sorting system for multiple bit binary records (Aug 27 1968), uS Patent 3,399,383
3. Batcher, K.E.: Sorting networks and their applications. In: Proceedings of the April 30–May 2, 1968, spring joint computer conference. pp. 307–314. AFIPS '68 (Spring), ACM, New York, NY, USA (1968)
4. Bose, R.C., Nelson, R.J.: A sorting problem. J. ACM 9(2), 282–296 (1962)
5. Choi, S.S., Moon, B.R.: A graph-based lamarckian-baldwinian hybrid for the sorting network problem. IEEE Trans. Evolutionary Computation 9(1), 105–114 (2005)
6. Chung, M.J., Ravikumar, B.: Bounds on the size of test sets for sorting and related networks. In: ICPP. pp. 745–751 (1987)
7. Floyd, R.W.: Algorithm 245: Treesort. Commun. ACM 7(12), 701– (Dec 1964)
8. Graham, L., Oppacher, F.: Symmetric comparator pairs in the initialization of genetic algorithm populations for sorting networks. In: Evolutionary Computation, 2006. CEC 2006. IEEE Congress on. pp. 2845 –2850 (0-0 2006)
9. Juille, H.: Evolution of non-deterministic incremental algorithms as a new approach for search in state spaces (1995)
10. Kennedy, J., Eberhart, R.: Particle swarm optimization. vol. 4, pp. 1942–1948 vol.4 (1995)
11. Knuth, D.E.: The Art of Computer Programming, Volume III: Sorting and Searching, 2nd Edition. Addison-Wesley (1998)
12. Koza, J.R.: Genetic programming: On the programming of computers by means of natural selection. Statistics and Computing 4(2) (1994)
13. Parberry, I.: Single-exception sorting networks and the computational complexity of optimal sorting network verification. Mathematical Systems Theory 23(2), 81–93 (1990)
14. Parberry, I.: Parallel Complexity and Neural Networks. MIT Press (1994)
15. R., S.: Analysis of shellsort and related algorithms. In: ESA '96: Fourth Annual European Symposium on Algorithms. pp. 25–27. Springer (1996)

16. Sherenaz W. Al-Haj Baddar, K.E.B.: Designing Sorting Networks: A new Paradigm. Springer (2011)
17. van Voorhis, D.C.: Toward a lower bound for sorting networks. In: Complexity of Computer Computations. pp. 119–129 (1972)
18. Waksman, A.: A permutation network. J. ACM 15(1), 159–163 (Jan 1968)

Simulación de grandes multitudes con dinámica de grupos

Oriam De Gyves[1], Leonel Toledo[1], Iván Rivalcoba[1], Isaac Rudomín[2]

[1] Tecnológico de Monterrey, Campus Estado de México,
Estado de México, México

[2] Barcelona Supercomputing Center,
Barcelona, España

Resumen. La simulación de multitudes ha sido utilizada para crear ambientes urbanos vibrantes e inmersivos, en donde miles agentes virtuales se comportan como peatones reales. Muchos investigadores se han enfocado en simulaciones constituidas únicamente de agentes individuales, sin embargo, en la realidad las multitudes están conformadas no solo de individuos, sino también de pequeños grupos de peatones que viajan juntos. En este trabajo presentamos un sistema que permite simular grandes multitudes, en el hardware gráfico, que además integra la noción de formación de pequeños grupos de agentes. Utilizamos una modificación de la técnica de fuerzas sociales para que los agentes naveguen en el ambiente virtual. Identificamos en ésta un error de formulación que ocasiona que los agentes se atoren bajo circunstancias específicas y proponemos una solución. Los resultados experimentales muestran un cambio en la dinámica de la multitud, y el sistema completo es capaz de simular grandes multitudes de manera interactiva.

Palabras clave: Simulación, grandes multitudes, dinámica de grupos.

1. Introducción

Planeación urbana, educación, seguridad y entretenimiento, entre otras, son algunas de las aplicaciones de la simulación de multitudes hoy en día [3]. Aplicaciones de este tipo pueden utilizarse en lugares públicos para detectar el comportamiento anormal de la población, como en un partido de fútbol o en el sistema subterráneo de una ciudad. Por otro lado, la industria del cine las ha utilizado para crear escenas de grandes batallas o invasiones. Este trabajo se centra en aplicaciones que requieren simulaciones en tiempo real, como el primer ejemplo, por lo que es importante la optimización de los algoritmos utilizados.

Esta área ha sido estudiada por diversos investigadores en la actualidad, sin embargo, es sólo recientemente que se comenzaron a estudiar los grupos que se forman dentro de una multitud. Las simulaciones microscópicas son aquellas que se enfocan en estudiar el comportamiento de los individuos dentro de una multitud. Tradicionalmente, este tipo de simulaciones ha considerado únicamente a los peatones de manera aislada, es decir, que no tienen ningún tipo de relación con algún otro peatón de la multitud. En la realidad, una multitud

está compuesta en su mayoría por pequeños grupos de peatones con algún tipo de relación, como familiares o amigos. Estos pequeños grupos mantienen ciertas formaciones que facilitan la interacción social, lo que cambia la dinámica de una multitud.

Para poder manejar grandes cantidades de agentes, la simulación de los agentes es realizada en el procesador gráfico. Para que el GPU (*Graphics Processing Unit*, unidad de procesamiento gráfico en inglés) pueda trabajar de una manera eficiente, es importante reducir la cantidad de información que se comunica entre éste y el CPU (*Central Processing Unit*, unidad de procesamiento central en inglés), ya que esa comunicación sería por cada cuadro de la simulación. Es por esta razón que utilizamos un método para realizar las búsquedas de proximidad en el GPU, además de los comportamientos, logrando así que toda la simulación se realice en paralelo sin necesidad de compartir información con el CPU por cada cuadro.

La contribución principal de este artículo es el presentar un sistema multiagente capaz de simular grandes multitudes a través de algoritmos altamente paralelos que usan de manera eficiente el procesador gráfico. Se hace uso de técnicas que permiten la simulación de multitudes con la formación de pequeños grupos de agentes virtuales relacionados. Los resultados experimentales indican que la dinámica de la multitud se ve afectada al incluir estas características. La inclusión de estas características podría significar una representación más acertada de la realidad, ayudando a otras áreas a crear aplicaciones más realistas.

El resto del artículo se divide de la siguiente manera. Se presenta una breve discusión sobre el trabajo relacionado en la Sección 2. En la Sección 3 se presenta una técnica diseñada para mejorar el desempeño de las búsquedas de proximidad en tiempo real, lo que permite simulaciones de grandes multitudes. La Sección 4 presenta nuestra formulación para incluir grupos de agentes virtuales en las simulaciones, así como una modificación a la formulación un algoritmo conocido para simulaciones microscópicas. Los experimentos que realizamos para probar nuestro sistema y los resultados son presentados en la Sección 5. Finalmente, se presentan las conclusiones, limitantes y el trabajo futuro en la Sección 6.

2. Trabajo relacionado

La simulación de multitudes es un área que ha sido estudiada de manera extensiva, tanto de manera macroscópica como microscópica. Este trabajo se enfoca principalmente en simulaciones microscópicas, ya que éstas se encargan del estudio de los comportamientos de los agentes, por lo que son más apegadas a la realidad. Esta Sección presenta una breve discusión el trabajo relacionado y del estado del arte.

Craig Reynolds [14] presentó uno de los primeros trabajos de investigación enfocado en comportamientos de agentes virtuales autónomos, llamados *boids*. Este trabajo se encargaba de la navegación de los agentes en base a reglas simples, llamadas *steering behaviors*, y Reynolds las presenta como bloques de

construcción para crear comportamientos complejos. El mismo autor extiende su investigación con comportamientos más complejos [15].

Helbing et al. [10] presenta otro modelo importante para la navegación de los peatones. Este modelo está basado en fuerzas, conocidas como *Fuerzas Sociales*, las cuales hacen referencia a las intenciones de un agente para realizar una acción. De acuerdo con los autores, este modelo es adecuado para peatones en situaciones normales o conocidas. Sin embargo, los mismos autores extienden su investigación para crear un modelo que también es adecuado para situaciones de estrés, al añadir un parámetro de nerviosismo que hace que el comportamiento de los peatones sea más errático y exhiba características de manada [9]. Moussaid et al. [13] extiende el concepto de Fuerzas Sociales para incluir comportamientos y formaciones de grupo entre los agentes. Los autores hacen un estudio sobre el tamaño común de los grupos en multitudes de diferente densidad y cómo éstos afectan la dinámica de la multitud. Millán et al. [12] presenta otra técnica basada en fuerzas, sin embargo, los autores codifican las fuerzas en texturas para poder implementarlo de manera eficiente en el GPU.

De igual manera, existen métodos de navegación geométricos, como los *Velocity Obstacles* introducidos por Fiorini et al. [5]. En este método, los agentes calculan el conjunto de velocidades que los llevaran a una colisión con un obstáculo; para moverse en rutas sin colisiones, los agentes eligen velocidades fuera de este conjunto. Este concepto es aumentado en *Reciprocal Velocity Obstacles* (RVO) por Van den Berg et al. [2] al dividir el esfuerzo de la navegación entre los agentes involucrados. El mismo grupo de investigación optimizó RVO a problemas de programación lineal de baja dimensión, lo que les permitió simular miles de agentes en tiempo real [1].

En todas las técnicas antes mencionadas, existen interacciones entre los peatones, independientemente de si pertenecen o no a un grupo dentro de la multitud. Existen estructuras jerárquicas, como los *octrees*, que subdividen el espacio en regiones que contienen agentes [7]. Una estructura jerárquica muy usada por los investigadores [2,6] es conocida como kd-trees. Recientemente, Dos Santos et al. [17] ha investigado la implementación de estas estructuras en el GPU. Hoff et al. [11] presenta una técnica de detección de colisiones geométrica, rasterizando imágenes y tratando pixeles sobre-escritos como colisiones. De Gyves et al. [4] presenta una técnica basada en diagramas de Voronoi truncado para realizar búsquedas de proximidad de manera eficiente en el GPU. Esta técnica es capaz de superar en desempeño a otras similares, resultando en simulaciones con miles de agentes en tiempo real, utilizando hardware de nivel consumidor.

3. Búsquedas de proximidad

Para poder simular de manera eficiente grandes multitudes, no basta con que la simulación de los comportamientos sea realizada en el GPU, aun cuando se realice en paralelo. Es importante no compartir grandes cantidades de información entre el CPU y el GPU, ya que esto puede afectar de manera negativa el desempeño de una aplicación. En simulación de multitudes, se trabaja

típicamente con miles de agentes virtuales, sin embargo, compartir información entre GPU y CPU en cada cuadro de la simulación para esta cantidad de agentes resulta prohibitiva. Debido a esto, no solo realizamos la simulación de los comportamientos en el GPU, también las búsquedas de proximidad. De esta manera se minimiza la comunicación entre el hardware gráfico y la unidad de procesamiento, resultando en un aumento en el desempeño.

En esta fase del sistema, utilizamos la técnica de diagramas de Voronoi truncado de De Gyves et al. [4]. Esta técnica demostró ser eficiente para el cálculo de búsquedas de proximidad en el GPU, aun cuando este sea considerado como *entry-level* (nivel básico). Dado un conjunto S de sitios en el espacio, un diagrama de Voronoi VD(S) es la descomposición de $\mathbb{R}^d$ en $|S|$ celdas de Voronoi, en donde:

$$V(s,S) = \{p \in \mathbb{R}^d \mid d(p,s) \leq d(p,s') \forall s' \in S\} \tag{1}$$

Por otro lado, un diagrama de Voronoi truncado tVD(S) es la descomposición de $\mathbb{R}^2$ en $|S|$ celdas de Voronoi truncado, en donde:

$$tV(s,S) = \{p \in \mathbb{R}^2 \mid d(p,s) \leq d(p,s') \ \wedge \ d(p,s) \leq r \forall s' \in S\} \tag{2}$$

La Ecuación 1 describe una celda de Voronoi similar a la de la Ecuación 2, con la diferencia de que las celdas en la segunda ecuación están acotadas por un radio r (llamado radio de celda) y no por celdas adyacentes. De esta manera, si la separación entre celdas es mayor a r, el diagrama de Voronoi truncado está compuesto de $|S|$ círculos. Por otro lado, si la separación entre todas las celdas es menor a r, no existe diferencia entre los diagramas.

Al usar estos diagramas, cada una de las celdas representa un agente en la simulación; el diagrama completo representa un mapa de ambiente que cada agente puede usar para obtener, por medio de una búsqueda local, a sus vecinos más cercanos. Se utilizan este tipo de diagramas, en lugar de dibujado tradicional, para que no existan conflictos con el orden de dibujado de las celdas, como se observa en la Figura 1. De igual manera, se garantiza que en caso de que la distancia entre 2 agentes sea menor a r, el área entre estos será distribuida de manera uniforme entre ambos, lo cual es útil para el proceso de muestreo del diagrama.

Para detectar a los vecinos más cercanos usando el mapa de ambiente, cada agente realiza un muestreo en la dirección de su movimiento. Cada agente a realiza un proceso conocido como *Ray Marching* en el diagrama. Comenzando en la posición de a, cada agente lanza n rayos en la dirección de su movimiento, distribuidos de manera uniforme en el área de visión del agente. Cada rayo realiza un muestreo en el mapa de ambiente a ciertos intervalos predeterminados (típicamente menor a r), como se muestra en la Figura 2; en cada muestreo se detecta si en ese punto hay algún otro agente para construir la lista de vecinos.

De Gyves et al. [4] hace un estudio sobre diferentes técnicas para calcular el diagrama. En este trabajo presentamos los algoritmos para generar el diagrama de Voronoi truncado utilizando el procesador gráfico. Como pre-procesamiento se prepara una textura con un gradiente radial. Esta textura es enviada al GPU

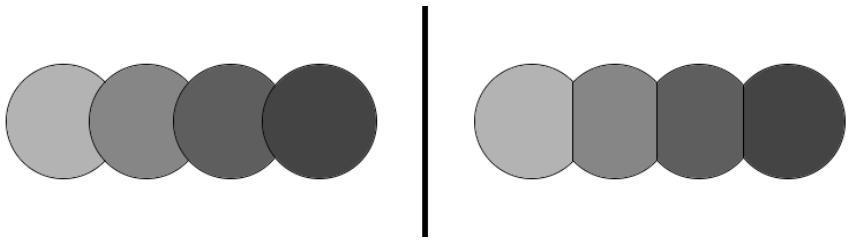


Fig. 1. Cuatro círculos representando a cuatro agentes, dibujados de izquierda a derecha. La imagen de la derecha utiliza diagramas de Voronoi truncado para dibujar las celdas.

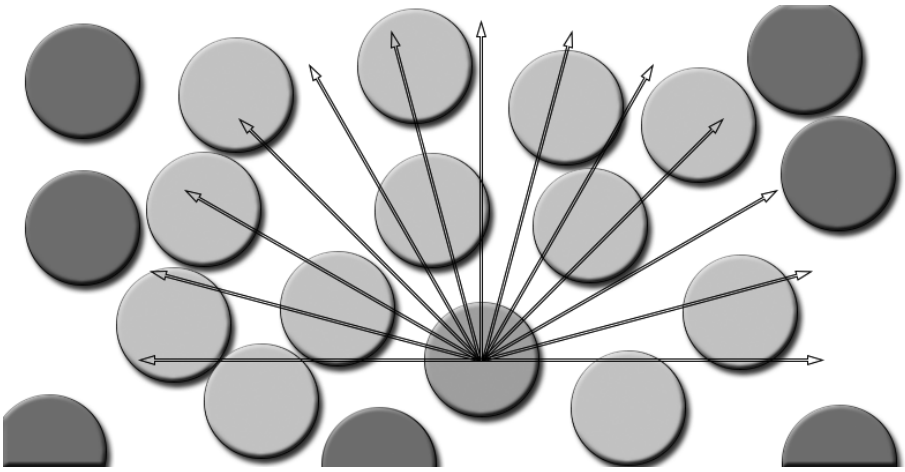


Fig. 2. Muestreo del mapa de ambiente lanzando rayos desde la posición de cada agente en la dirección de su movimiento

y rasterizada en cada celda de Voronoi truncado. La construcción de la textura es realizada en CPU y el Algoritmo 1 presenta los pasos para generarla. Una vez generada la textura, ésta se envía al GPU para poder dibujar el diagrama de Voronoi truncado.

Una vez que la textura con gradiente radial haya sido enviada al GPU, los valores del gradiente se usan como una representación de profundidad del fragmento a rasterizar. De esta manera se utiliza el *Z-buffer* (técnica implementada en todas las tarjetas gráficas actuales para prevenir *Z-fighting*) como un método eficiente para descartar los fragmentos de celdas superpuestas. La implementación de esta técnica es completamente realizada en el GPU, utilizando OpenGL y *GLSL*. El Algoritmo 2 muestra como dibujar cada celda de Voronoi truncado, utilizando una función de prueba de profundidad *GL_LESS*.

Algoritmo 1 Pre-procesamiento. Creación de una textura con un gradiente radial que es utilizada por cada celda de Voronoi truncado.

Requiere: d hace referencia al diámetro de la textura con el gradiente radial.

```
1: función CrearTextura(d)                    ▷ Creación de una textura de diámetro d
2:     tex[d] ← [0, 0, ..., 0]
3:     r ← d/2
4:     de i ← 0 a d realiza
5:         de j ← 0 a d realiza
6:             v ← i − R
7:             h ← j − R
8:             dst ← sqrt(v^2 + h^2)
9:             si dst > r entonces
10:                dst ← r
11:            tex[i × d + j] ← 1 − dst/r
12:    return tex
13: fin función
```

4. Dinámica de grupos

Recientemente han surgido investigaciones enfocadas en incluir grupos de peatones en simulaciones de multitudes. Tal es el caso de Moussaid et al. [13], quien presenta uno de los primeros trabajos que incluye una formulación para crear grupos de peatones dentro de una multitud. Los autores realizan además un estudio sobre las formaciones de los peatones, dependiendo de la densidad de peatones en el área. De acuerdo con sus resultados, los peatones normalmente viajan en formaciones tipo-V, lo que facilita la comunicación e interacción social entre los miembros del grupo.

Algoritmo 2 Dibujado del diagrama de Voronoi truncado utilizando GLSL. Este procedimiento se realiza por cada fragmento a rasterizar de cada celda de Voronoi truncado.

Requiere: tex hace referencia a una textura con gradiente radial. x y y hacen referencia a las coordenadas del fragmento que se va a rasterizar (x, y). c representa el color a dibujar en el fragmento.

```
1: procedimiento DibujarFragmento(tex, c, x, y)   ▷ Ejecutado una vez por cada fragmento.
2:     depth ← 1 − fetch(tex, x, y)               ▷ fetch lee el valor de tex en (x, y)
3:     si depth < f_d entonces   ▷ f_d es el valor de profundidad actual del fragmento
4:         fc = c                                 ▷ f_c es el color actual del fragmento
5:         f_d = depth
6: fin procedimiento
```

Realizamos tomas aéreas de estudiantes caminando en el pasillo de una universidad de la Ciudad de México, como se puede observar en la Figura 3. Los videos fueron grabados en 6 momentos distintos durante un día normal de clases en la universidad. En este estudio observamos a más de 150 personas caminando en varias direcciones, de las cuales el 57 % pertenecía a un pequeño grupo. De acuerdo con Moussaid, hasta el 70 % de los peatones que ellos observaron caminaban en grupos. En ambos casos, la proporción de peatones que viajan en grupos es mayor a la de peatones que viajan solos. De igual manera, en nuestro estudio podemos observar que los grupos de dos a cuatro miembros son los más comunes, mientras que grupos de mayor tamaño se vuelven raros. En nuestro estudio, solo pudimos observar un grupo de más de 4 personas.

Fig. 3. Cinco imágenes de peatones caminando por un pasillo. Como se puede observar, la mayoría de los peatones no camina solo, sino en compañía de otros peatones.

Para que nuestros agentes naveguen por un ambiente virtual, realizamos una implementación en paralelo del trabajo de Moussaid et al. [13]. Como en el caso de las búsquedas de proximidad, los comportamientos de los agentes son implementados en el GPU, utilizando OpenGL y GLSL. Sin embargo, en el caso de los comportamientos, fueron implementados utilizando *Compute Shaders*. Este tipo de shaders permite el uso de cómputo de propósito general utilizando

unidades de procesamiento gráfico (GPGPU o *General Purpose Computations on Graphics Processing Units*), además de permitir el uso de elementos gráficos (la textura con el mapa de ambiente). Rivalcoba et al. [16] ha utilizado una técnica similar para acoplar una multitud virtual con peatones reales capturados en video.

Utilizamos cuatro texturas con los datos requeridos para la simulación: posición, velocidad, grupo y metas. Cada elemento de la texturas (*texel*) i contiene los datos del agente a_i. Para actualizar los valores de las texturas de posición y metas, utilizamos las Ecuaciones 3, 4 y 5.

$$\boldsymbol{f}_i = \boldsymbol{f}_i^d + \boldsymbol{f}_i^n + \boldsymbol{f}_i^o + \boldsymbol{f}_i^g \tag{3}$$

En donde $\boldsymbol{f}_i$ representa la aceleración del agente a_i en el cuadro actual. Esta fuerza es modificada por las motivaciones internas del agente ($\boldsymbol{f}_i^d$), sus vecinos más cercanos ($\boldsymbol{f}_i^n$), los obstáculos más cercanos ($\boldsymbol{f_i}^o$) y la dinámica de grupo ($\boldsymbol{f}_i^g$).

$$\boldsymbol{v}_{inew} = \boldsymbol{v}_i + \boldsymbol{f}_i \cdot \Delta t \tag{4}$$

La nueva velocidad es representada por $\boldsymbol{v}_{inew}$, y se obtiene al sumar la aceleración ($\boldsymbol{f}_i$) con la velocidad anterior ($\boldsymbol{v}_i$).

$$\boldsymbol{p}_{inew} = \boldsymbol{p}_i + \boldsymbol{v}_{inew} \cdot \Delta t + \frac{\boldsymbol{f}_i \cdot \Delta t^2}{2} \tag{5}$$

Finalmente, la posición del agente ($\boldsymbol{p}_{inew}$) se actualiza tomando en cuenta su aceleración y velocidad. En las Ecuaciones 4 y 5, Δt representa el tiempo transcurrido entre dos cuadros de la simulación consecutivos, y se utiliza normalmente en aplicaciones gráficas para obtener velocidades de movimientos independientes del *frame-rate*. Por otro lado, la dinámica de grupo puede ser expresada de la siguiente manera:

$$\boldsymbol{f}_i^g = \boldsymbol{f}^v + \boldsymbol{f}^a + \boldsymbol{f}^r \tag{6}$$

En donde $\boldsymbol{f}^v$ representa la visibilidad del agente. Normalmente se prefiere que los otros miembros del grupo permanezcan dentro del rango de visión del agente. $\boldsymbol{f}^a$ representa la atracción hacia otros miembros de su grupo, representado por el centro de masa del grupo. Finalmente, la repulsión hacia otros agentes, sean o no de su grupo, es representada por $\boldsymbol{f}^r$.

El trabajo de Moussaid está basado en Fuerzas Sociales de Helbing [8]. En este trabajo se describe una *fuerza de cuerpo* y una *fuerza de deslizamiento*, las cuales contrarrestan la compresión de los cuerpos y generan un movimiento relativo tangencial, respectivamente. Al implementar el método, encontramos que la diferencia en la velocidad tangencial ($\Delta \boldsymbol{v}_{ji}^t = (\boldsymbol{v}_j - \boldsymbol{v}_i) \cdot t_{ij}$) usada en la fuerza de deslizamiento es cero ($\Delta \boldsymbol{v}_{ji}^t = 0$) cuando los agentes se mueven en direcciones opuestas con la misma velocidad. Esto resulta en agentes que no pueden evadirse, dejándolos atorados en una línea recta, como puede observarse en la Figura 4.

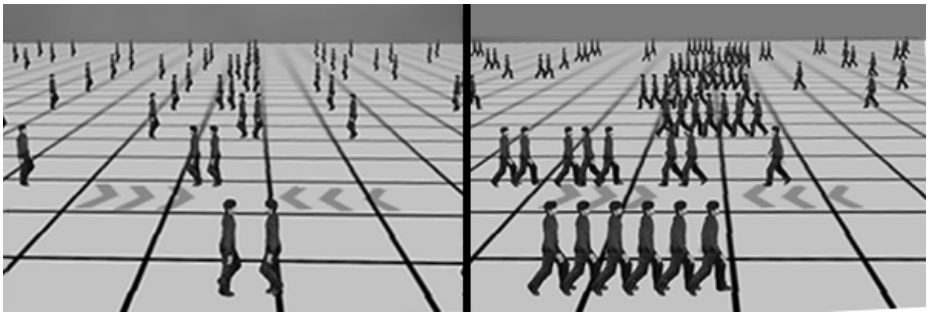


Fig. 4. Los agentes se quedan atorados en una línea recta utilizando la formulación original de Fuerzas Sociales

$$\Delta \boldsymbol{v}_{ji} = \upsilon \tag{7}$$

En la Ecuación 7, υ representa una constante vectorial que satisface $\|\upsilon\| \neq 0$ y es usada en caso de que $\Delta \boldsymbol{v}_{ji} = \boldsymbol{v}_j - \boldsymbol{v}_i \leq \epsilon$. La magnitud de υ es proporcional a la fuerza de deslizamiento, es decir, a mayor fuerza de deslizamiento los agentes se evaden más rápidamente. La prueba anterior fue repetida utilizando esta corrección en la formulación original de la diferencia de velocidad tangencial, con buenos resultados, como se observa en la Figura 5.

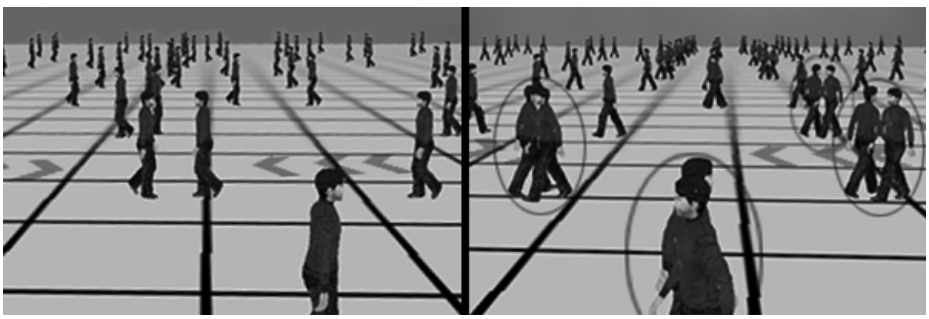


Fig. 5. Con los cambios en la formulación en la diferencia de velocidad tangencial, los agentes pueden evadirse aún bajo las mismas circunstancias que los atoran en una línea recta usando la formulación original de Fuerzas Sociales

5. Experimentos y resultados

Los escenarios que se describen a continuación, así como las técnicas presentadas anteriormente, fueron implementados usando C++ para el código de CPU y OpenGL/GLSL para la simulación y visualización de la multitud en el GPU. Las características de hardware son las siguientes: computadora personal x64 con un procesador Intel i7-2630QM de cuatro núcleos a 2.00 GHz y 8GB de memoria. La tarjeta de video es una nVidia GT540M, la cual es de la familia Fermi y soporta OpenGL 4.3. Esta tarjeta de video es considerada de nivel básico para los estándares actuales.

Realizamos pruebas de evasión de colisiones para verificar que nuestra formulación conservara una correcta navegación de los agentes por el entorno virtual. Esta prueba es conocida como Circle-N (Figura 6), en donde N agentes son colocados en la circunferencia de un círculo y su meta es llegar a la posición diametralmente opuesta. A pesar de que esta prueba no refleja cómo se mueven los peatones en la realidad, es usualmente empleada para probar la navegación de los agentes en una multitud. En esta prueba no se incluye navegación global por la misma razón.

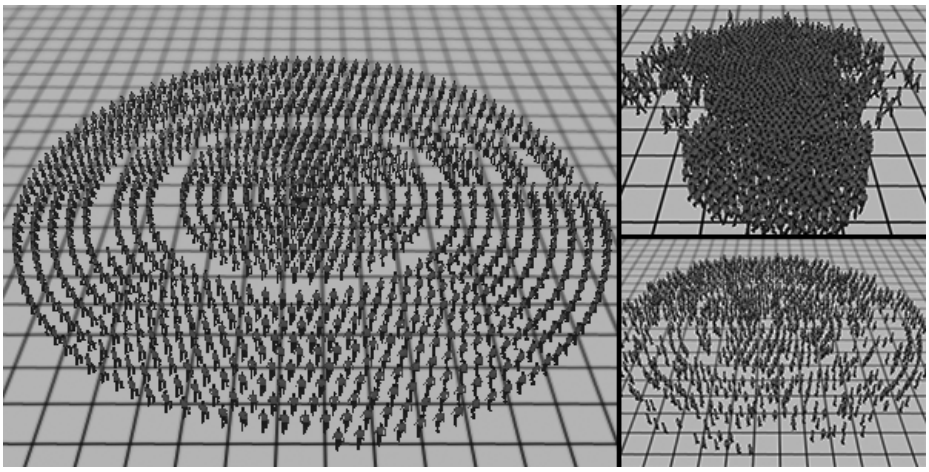


Fig. 6. Prueba de evasión de colisiones. Los agentes inician en la circunferencia de círculos concéntricos y su meta se encuentra en la posición diametralmente opuesta.

El sistema debe ser capaz de simular grandes multitudes (varios miles de agentes) en tiempo real. En la Figura 7 se muestran dos pruebas con hasta 16 mil agentes, simuladas a 60 cuadros por segundo. En estas pruebas se puede observar que la velocidad promedio a la que viajan los agentes, en los escenarios con grupos, disminuye más de 10 % en comparación con los escenarios sin esta característica (de 1.21 m/s a 1.085 m/s). Debido a esta disminución en la velo-

cidad promedio de la población, los agentes tardan más tiempo en llegar a sus metas, ya sea que viajen en grupo o no. Esto refleja el cambio en la dinámica de la multitud cuando se agregan este tipo de propiedades.

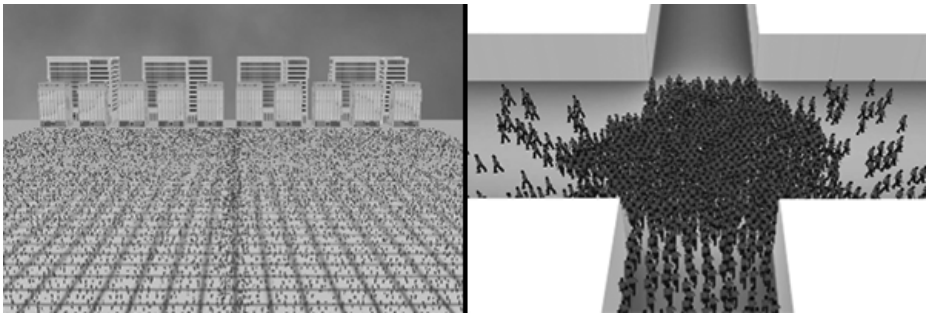


Fig. 7. Pruebas con hasta 16,384 agentes virtuales. Simulaciones en tiempo real (16.6 ms. por cuadro).

La Figura 8 muestra los grupos que se forman en nuestras simulaciones, en donde se han resaltado algunos de los grupos. Con un 4 % de probabilidad, los grupos pueden estar formados por 4 miembros, 12 % por 3 miembros, 40 % por 2 miembros y el resto de los agentes viaja de manera individual. De esta manera, en promedio las simulaciones tienen un mayor número de agentes que viajan en grupo sin que todas las simulaciones tengan las mismas características. Realizamos simulaciones desde 1,024 hasta 16,384 agentes virtuales en tiempo real.

6. Conclusiones, limitantes y trabajo futuro

Las multitudes no son constituidas únicamente de individuos viajando solos; la mayor parte de una multitud se conforma de pequeños grupos de peatones relacionados. Realizamos un estudio para confirmar que la proporción de pequeños grupos normalmente es mayor a los individuos que viajan solos, obteniendo los resultados esperados. Hemos presentado un sistema capaz de simular grandes multitudes que también considera pequeños grupos de agentes relacionados. Este sistema es paralelo e implementado en el hardware gráfico utilizando técnicas de rasterizado y de cómputo general en GPUs. De esta manera se han conseguido simulaciones con miles de agentes en tiempo real.

El sistema de búsquedas de proximidad depende de una textura de ambiente, el diagrama de Voronoi truncado. Este diagrama representa el mundo virtual por donde se mueven los agentes, sin embargo, mientras más grande sea el mundo, más grande debe ser el tamaño de la textura. Esto significa que existe un impacto negativo en el desempeño del sistema conforme mayor sea el tamaño del

Fig. 8. En las imágenes se muestran algunos de los grupos que se forman en nuestro sistema. Los agentes intentan viajar juntos hasta su meta.

mundo. Como trabajo futuro, investigaremos el uso de técnicas de nivel de detalle (específicamente utilizando *mipmaps*) para hacer independiente el tamaño de la textura del tamaño del mundo virtual.

Por otro lado, el sistema de navegación de los agentes está basado en fuerzas. Este tipo de sistemas sufren de problemas cuando las fuerzas de varios agentes se cancelan unas con otras, ocasionando que los agentes no evadan colisiones adecuadamente. Buscaremos trasladar este sistema a un método geométrico, para evitar que sucedan estas posibles fallas. De igual manera, continuaremos investigando la asignación de roles a los miembros de un grupo, como *mamá, amigo, novio, hermana*. La introducción de estos roles podría cambiar también la dinámica de movimiento de una multitud, y acercaría cada vez más estas simulaciones a la realidad.

Referencias

1. van den Berg, J., Guy, S.J., Lin, M., Manocha, D.: Reciprocal n-body collision avoidance. Robotics Research 70, 3–19 (2011)
2. van den Berg, J., Manocha, D.: Reciprocal Velocity Obstacles for real-time multi-agent navigation. In: 2008 IEEE International Conference on Robotics and Automation. pp. 1928–1935. IEEE (May 2008)
3. De Gyves, O., Toledo, L., Rudomín, I.: Comportamientos en simulación de multitudes : revisión del estado del arte. Research in Computing Science. 62(Special Issue: Avances en Inteligencia Artificial.), 319–334 (2013)
4. De Gyves, O., Toledo, L., Rudomín, I.: Proximity Queries for Crowd Simulation Using Truncated Voronoi Diagrams. In: Proceedings of Motion on Games - MIG '13. pp. 87–92. ACM Press, New York, New York, USA (2013)

5. Fiorini, P., Shiller, Z.: Motion Planning in Dynamic Environments Using Velocity Obstacles. The International Journal of Robotics Research 17(7), 760–772 (Jul 1998)
6. Guy, S.J., Chhugani, J., Curtis, S., Dubey, P., Lin, M., Manocha, D.: PLEdestrians: a least-effort approach to crowd simulation. Proceedings of the 2010 ACM SIGGRAPH/Eurographics Symposium on Computer Animation pp. 119–128 (2010)
7. Hadap, S., Eberle, D., Volino, P., Lin, M.C., Redon, S., Ericson, C.: Collision detection and proximity queries. In: Proceedings of the conference on SIGGRAPH 2004 course notes - GRAPH '04. pp. 15–es. ACM Press, New York, New York, USA (2004)
8. Helbing, D., Farkas, I., Vicsek, T.: Simulating dynamical features of escape panic. Nature 407(6803), 487–90 (Sep 2000)
9. Helbing, D., Farkas, I.J., Molnár, P., Vicsek, T.: Simulation of Pedestrian Crowds in Normal and Evacuation Situations. Pedestrian and evacuation dynamics 21 (2002)
10. Helbing, D., Molnár, P.: Social force model for pedestrian dynamics. Physical Review E 51(5), 4282–4286 (May 1995)
11. Hoff, K.E., Zaferakis, A., Lin, M., Manocha, D.: Fast and simple 2D geometric proximity queries using graphics hardware. In: Proceedings of the 2001 symposium on Interactive 3D graphics - SI3D '01. vol. 31, pp. 145–148. ACM Press, New York, New York, USA (Feb 2001)
12. Millan, E., Hernandez, B., Rudomin, I.: Large Crowds of Autonomous Animated Characters Using Fragment Shaders and Level of Detail. In: Wolfgang Engel (ed.) ShaderX5: Advanced Rendering Techniques, chap. Beyond Pix, pp. 501—-510. Charles River Media (2006)
13. Moussaïd, M., Perozo, N., Garnier, S., Helbing, D., Theraulaz, G.: The walking behaviour of pedestrian social groups and its impact on crowd dynamics. PloS one 5(4), e10047 (Jan 2010)
14. Reynolds, C.W.: Flocks, herds and schools: A distributed behavioral model. ACM SIGGRAPH Computer Graphics 21(4), 25–34 (Aug 1987)
15. Reynolds, C.W.: Steering behaviors for autonomous characters. In: 1999 Game Developers Conference. pp. 763–782 (1999)
16. Rivalcoba Rivas, J.I., De Gyves, O., Pelechano, N., Rudomín, I.: Coupling Camera-tracked Humans with a Simulated Virtual Crowd. In: Proceedings of the 9th International Conference on Computer Graphics Theory and Applications. pp. 312–321. SCITEPRESS - Science and and Technology Publications (2014)
17. dos Santos, A.L., Teixeira, J.M.X., de Farias, T.S., Teichrieb, V., Kelner, J.: kD-Tree Traversal Implementations for Ray Tracing on Massive Multiprocessors: A Comparative Study. In: 2009 21st International Symposium on Computer Architecture and High Performance Computing. pp. 41–48. IEEE (Oct 2009)

Aceleración de la velocidad de procesamiento a través de la paralelización de algoritmos

Israel Tabarez Paz[1], Neil Hernández Gress[2], Miguel González Mendoza[2]

[1] Universidad Autónoma del Estado de México,
Atizapán de Zaragoza, México

Universidad Politécnica del Valle de Toluca,
Estado de México, México

[2] Tecnológico de Monterrey, Campus Estado de México,
Atizapán de Zaragora, México

itabarezp@uaemex.mx, {ngress, mgonza}@itesm.mx

Resumen. En este artículo se analizarán los aspectos más importantes para la paralelización de algoritmos. Por lo que nuestro interés sobre el tema es debido a sus múltiples campos de aplicación con el propósito principal es acelerar la velocidad de procesamiento para la solución de problemas tengan un alto costo computacional. Su principal funcionalidad es reducir el tiempo de ejecución del programa a traés de la aplicación de técnicas, y bajo ciertas condiciones. Para llevar a cabo esta tarea es necesario diseñar un algoritmo para su programación y elegir una infraestructura de computación paralela. Una de las aplicaciones de los algoritmos paralelos es la optimización de las operaciones matriciales y vectoriales para clasificacin de grandes bases de datos.

Palabras clave: GPU, CUDA, algoritmos paralelos, velocidad de procesamiento, redes neuronales artificiales; máquinas de soporte vectorial.

1. Introducción

El presente trabajo está enfocado al análisis del desempeño de la paralelización de algoritmos para la aceleración de la velocidad de procesamiento, utilizando un CPU Intel Core 2 Duo, de 2.66GHz. La tarjeta gráfica aplicada es una NVIDIA GeForce 9400M. Sus principales problemáticas son: que tan paralelizables son los algoritmos, el tipo de arquitecturas que se pueden utilizar, las ventajas y desventajas de la aceleración del procesamiento. Para analizar estas problemáticas es necesaria la búsqueda o el estudio de diferentes alternativas ya sean métodos, herramientas y aplicaciones que puedan ayudar a resolverlas. El interés de esta investigación es conocer las ventajas y desventajas de la paralelización para sistemas distribuidos con la mayor precisin posible. Este artículo está distribuido en las siguientes secciones: estado del arte en la sección 2, experimentación en la sección 3 y conclusiones en la sección 4.

2. Estado del arte

El paralelismo es una forma en la cual pueden realizarse varios cálculos simultáneamente. Está basado el en principio de dividir problemas grandes para obtener problemas pequeños los cuales posteriormente son solucionados en paralelo.

La programación paralela permite:

- Resolver problemas con gran cantidad de datos.
- Reducción del tiempo de solución.

Pereira [1], describe el paralelismo sobre conjuntos de datos métricos y un algoritmo de búsqueda por similitud basado en una estructura de indexación, utilizando el modelo de filtros. Dicho algoritmo mostró un desempenño eficiente en la praáctica respecto del número de procesadores disponibles, en redes de tamano moderado.

Aracena [3], presenta una optimización del método de detección de puntos SIFT (Scale Invariant Feature Transform), mediante su paralelización empleando una GPU (Unidad de Procesamiento Gráfico). El objetivo es acelerar el tiempo de cómputo, para la detección de puntos característicos. Asimismo, el autor indica que se basa en dos premisas: el balance carga y la distribución de cálculo. Respecto a las pruebas se realizaron en con procesador Core 2 Duo 2.2Ghz, 3GB de RAM y una VGA GeForce 8600GT (32 núcleos) de 512MB. Respecto a los resultados nos indica que se logró un rendimiento de 42.5 milisegundos en promedio. Las pruebas fueron realizadas en diferentes resoluciones e indica que la paralelización de SIFT no muestra pérdidas significativas de puntos claves en comparación con la versión secuencial.

Armenteros [4], estudia diferentes alternativas para la aceleración y muestra su efecto en el análisis de metaheurísticas paralelas, asimismo, ofrece una guía de cómo y cuándo utilizarlas. Finalmente, concluye que el mejor escenario para la aceleración es definir la calidad de las soluciones como condición de parada, porque todas las ejecuciones alcanzan el óptimo y permite una comparación justa entre los métodos.

Uribe [5], presenta la implementación de una estructura genérica adaptada para un sistema multiprocesador con una GPU, mostrando los resultados experimentales en trminos de tiempo y speed-up (aceleración). También, muestra experimentalmente las ventajas comparativas al insertar GPU a una plataforma multicore, así como el análisis del consumo energético.

Hidrobo [6], presenta un Sistema Manejador de Ambientes Reconfigurables para procesamiento paralelo/Distribuido. Asimismo, presenta dos enfoques, en el primero el programa se adapta al sistema y en el segundo sigue el procedimiento inverso. El autor se basa en la teoría de grafos para resolver problemas NP completos. También utiliza Algoritmos Genéticos técnica para encontrar una solución.

Dally [8], implementa la paralelización de redes de comunicación VLSI (integración a muy larga escala) analizando los costos para la realización lo cual

está en función de la arquitectura de la red. Asimismo, analiza el rendimiento de la red.

Duato [9], trabaja sobre la reconfiguración dinámica de la interconexión de la red para reducir la sobrecarga de comunicación. El trabajo se centra en la reconfiguración dinámica, es decir, la topología de red para cambiarla arbitrariamente en tiempo de ejecución. En este trabajo, se exponen en profundidad el algoritmo de reconfiguración y los diferentes conceptos relacionados con ella.

Flynn [10], presenta un modelo jerárquico de las organizaciones de informática en base a un modelo de árbol utilizando recursos de tipo petición / servicio como nodos. Se distinguen dos aspectos del modelo : lógica y física . Las organizaciones generales paralelas y múltiple son examinadas en cuanto a tipo y efectividad. El proceso simplex superpuesto (SISD) está limitado por las dependencias de los datos . La ramificación tiene un efecto particularmente degenerativa. Se analizan los procesadores paralelos [stream –data – stream mltiple – una sola instrucción (SIMD)]. Asimismo, se analiza el rendimiento de un procesador en paralelo, el cual aumenta logarítmicamente. También, los Multiprocesadores (MIMD) son sometidos a saturación basada en el bloqueo general de comunicaciones. De acuerdo con este autor, los modelos de colas simplificado indican que la saturación se desarrolla cuando la fracción de tiempo de la tarea (L/E) se aproxima a 1/n, donde n es el número de procesadores . Al compartir recursos en multiprocesadores se pueden utilizar para evitar otros problemas organizativos clásicos.

Tabla 1. Taxonomía Flynn.

	SIMPLE INTRUCCIÓN	MÚLTIPLES INSTRUCCÍONES
SIMPLES DATOS	SISD	SIMD
MÚLTIPLES DATOS	MISD	MIMD

De acuerdo con de acuerdo con Nikola [12], la taxonomía de paralelización se aproxima a la neurosimulaciones representadas en la figura 1.

Finalmente, es importante resaltar que no se hace un análisis de efectividad en function a la longitud de los datos, tampoco se indica el grado de paralelizacin de los algoritmos. Esto fue una motivación para realizar este trabajo.

2.1. Programación paralela

La programación paralela es el uso de varios procesadores trabajando simultáneamente juntos para resolver una tarea en común.

Existen diferentes formas [7]: nivel bit (microcontrolador), nivel instrucción o dato (GPU), y nivel tarea (como MPI). En la figura 2 podemos ver el principio del cómputo paralelo. Los diversos problemas pueden ser resueltos en lenguajes CUDA, OpenGL, Cg, C, C++ and FORTRAN.

Los principales aspectos que se consideran en la programación paralela son:

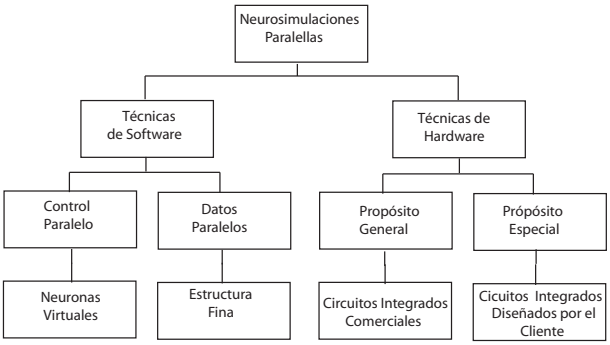


Fig. 1. Taxonomía de paralelización para neurosimulaciones.

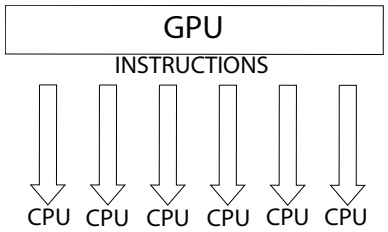


Fig. 2. Ejecución de instrucciones en paralelo.

- Arquitectura.
- Diseño del algoritmos.
- Evaluación de los algoritmos paralelos.

En la actualidad existen algunas opciones de hardware como FPGAs (Field Programmable Gate Array), GPUs (Graphics Processing Unit), a través del diseño de circuitos lógicos y secuenciales, microcontroladores, DSPs (Procesador Digital de Señales), PLCs (Controladores Lógicos Programables). En donde la mayoría de ellos trabajan el procesamiento paralelo a nivel bit, sin embargo los GPUs trabajan dicho procesamiento a nivel instrucción, en donde los diferentes núcleos ejecutan la misma instrucción al mismo tiempo, pero con diferente dato de entrada.

Respecto al diseño de algoritmos, es muy importante considerar que los algoritmos tienen grados de paralelización, lo cual está en función de la cantidad de instrucciones paralelizables. Finalmente, es necesario validar los resultados contrastando con los resultados esperados. En este trabajo, agregamos que la ecuación (1) nos permite calcular el porcentaje de paralelización de un algoritmo.

$$\%\ de\ Paralelizacion\ =\ \frac{Operaciones\ paralelizadas}{Total\ de\ operaciones\ seriales} \times 100 \qquad (1)$$

Aplicaciones de la programación paralela En el diseño de los algoritmos paralelos es importante el enfoque para solucionar problemas grandes para campos de aplicación. Por esta razón se determina que las etapas de diseño de algoritmos paralelos son:

- Particionamiento: Dividir en subproblemas.
- Comunicación: Sincronización de Tareas.
- Agrupación: Avaluación de la eficiencia y costos.
- Asignación de Tareas. Maximizar el uso de los recursos de los procesadores paralelos.

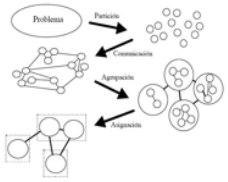


Fig. 3. Etapas de diseño de algoritmos paralelos.

En la figura (3) se ilustra cada etapa del diseño para la paralelización del algoritmo. Por otro lado, existe una gran cantidad de aplicaciones de paralelización de algoritmos como clasificación de grandes bases de datos a través de redes neuronales artificiales (RNA), lo cual nos permite reducir en gran medida la velocidad de procesamiento en un algoritmo que por naturaleza es secuencial y que consume gran cantidad de tiempo de procesamiento, pero con grandes oportunidades de paralelización.

Otras aplicaciones son la predicción a través de redes neuronales artificiales; implementación de algoritmos para tratamiento de imágenes; circuitos de sistemas de control con microcontroladores y procesadores digitales de señal; asimismo en la simulación de imágenes tridimensionales para diseño en ingeniería y simulación de fenómenos físicos, económicos, financieros y sociales, pero sobre todo de videojuegos, en donde se obtiene una gran capacidad de resolución de imágenes y precisión de la dinámica del movimiento, aunado a la gran velocidad de procesamiento. Esto nos permite tener una poderosa herramienta para el desarrollo de la investigación científica para el clculo de arreglos numéricos con dimensiones muy grandes.

Finalmente hay una gran cantidad de aplicaciones sin embargo, la programación paralela es diferente para cada arquitectura. En este caso analizaremos un caso de estudio aplicado a clasificación de bases de datos a través de redes neuronales artificiales de tercera generacin como son las Máquinas de Soporte Vectorial (SVM).

3. Máquinas de soporte vectorial (SVM)

En esta sección se presenta un caso aplicando paralelización de algoritmos que consiste en la simplificación del tiempo de ejecución de las operaciones matriciales y vectoriales del algoritmo SVM QP (Programación Cuadrática para las Máquinas de Soporte Vectorial). La ejecución se realizó sobre la plataforma de Mac OS X Lion 10.7.5, utilizando la tarjeta NVIDIA GeForce 9400M.

El GPU fue configurado con más de un bloque para cubrir todos los elementos de la base de datos. Lo que implicó la reducción de la paralelización a nivel de operaciones vectorial, además, de considerar todos los bloques posibles para las bases de datos con máximo 1400 instancias aproximadamente, y la paralelización óptima para 16 datos, ya que es el tamaño máximo en hilos de un bloque. Por otro lado, a menor paralelismo del código implicó aumentar la cantidad de funciones kernel adems de combinar código serial con paralelo, por lo que se requirió copiar del GPU al CPU varias veces y viceversa, debido a la naturaleza del código. Esto afectó de manera importante a la metodología SVM. En este caso para más de 1400 instancias se consideró a la metodología SVM light. El modelo matemático de SVM QP está descrito como:

$$\begin{gathered} \text{máximo } \alpha \\ q(\boldsymbol{\alpha}) = 0.5\boldsymbol{\alpha}^{\boldsymbol{T}} Q\boldsymbol{\alpha} - 1^T\alpha \qquad (2) \\ \text{sujeto a} \\ \boldsymbol{y}^T\boldsymbol{\alpha} = 0 \\ 0 \leq \boldsymbol{\alpha} \leq \boldsymbol{C} \end{gathered}$$

La matriz $\boldsymbol{Q}$ está compuesta por $(\boldsymbol{Q})_{ij} = y_i y_j k(x_i, x_j)$, donde:
$i, j = 1, 2, 3, ..., l,\ \boldsymbol{\alpha} = [a^1 ... a^l]^T$
$\mathbf{1} = [1^1 ... 1^l]^T\ \boldsymbol{y} = [y^1 ... y^l]^T$
$\boldsymbol{C} = [C^1 ... C^l]^T$
Finalmente, la salida es
$y = sign(\sum_{i=1}^{N} y_i \alpha_i K(\boldsymbol{x}, \boldsymbol{x_i}))$.
La figura 4 representa la arquitectura de SVM.

En este caso, fue paralelizada la multiplicación matricial para $((Q)_{ij} = y_i y_j k(x_i, x_j))$ en el método QP. El número de hilos fue generalizado como sigue:

$$Hilos\ por\ bloque = \frac{Cantidad\ de\ Datos}{Hilos\ por\ bloque} + 1 \qquad (3)$$

Una desventaja es que algunas veces las tarjetas gráficas tienen un número de hilos por bloque muy limitado, por lo que esa es la longitud máxima de un vector o número de datos copiados del CPU al GPU.

Es así que los resultados de SVM se muestran en la tabla 2. Para los casos mayores a 1400 instancias, se aplicó SVM light, y en los dems casos SVM QP, ya que una los recursos computacionales del GPU no fueron suficientes para hacer los cálculos correspondientes. Finalmente, en dicha tabla observamos los

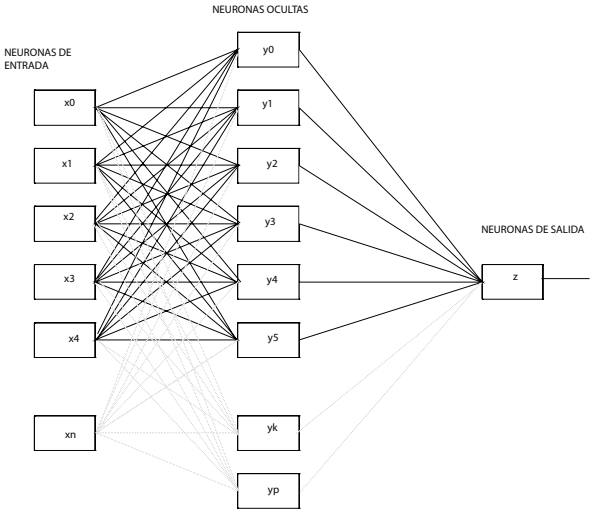


Fig. 4. Arquitectura de SVM.

resultados de paralelizar esta red neuronal SVM, en donde la clasificacin es binaria, como en el caso de la base de datos Cars, la cual consiste de 4 clases, cada una de ellas es clasificada de manera independiente.

Tabla 2. Paralelización del algoritmo SVM.

Datos	Instancias	Atributos	Clases	Iteraciones	Exitos	Tiempo de Aprendizale [ms]
Iris	150	4	3	2758	145	1521128.5
Cars (SVM light)	1728	6	4	———	1728 (Clase 1) 56 (Clase 2) 4 (Clase 3) 34 (Clase 4)	608927.125 1478488.5 1090742.125 970055.375
Breast Cancer Wisconsin	699	9	2	———	641 (Clase 1) 127 (Clase 2)	163790.55625 162216.453125
Adult	32561	14	2	———	———	———
Heart Disease Cleveland (HDC)	303	13	5	———	147 (Clase 1) 3 (Clase 2) 66 (Clase 3) 25 (Clase 4) 2 (Clase 5)	56749.73046 21624.41406 22715.71875 27424.23046 98908.94531

La cantidad mínima de instancias es 150 (Iris), y la cantidad mxima es 1728 (Cars). Respecto al tiempo, el mínimo fue registrado para Heart Disease Cleveland (HDC) para 303 instancias y 5 clases y el máximo para Cars con 4 clases diferentes.

Tabla 3. Resultados en Matlab.

Datos	Instancias	Clase	Tiempo de Aprendizale [s]
Iris	150	clase 1	0.0310
	50	clase 2	0.0000 (Total: 0.0310)
Cars	1728	clase 1	32.642
	1343	clase 2	18.391
	1275	clase 3	17.625 (Total: 68.658)
Breast Cancer Wisconsin	699	Calse 1	3.906
Hepatitis	155	Calse 1	1.516

En la tabla 3 se muestran los resultados aplicando MatLab, en donde observamos que el mínimo tiempo de procesamiento es de 0.0310 s, y el máximo es de 68.658 s. Comparando con los resultados obtenidos en la Tabla 2, el tiempo de procesamiento es menor en el CPU ya que el alto grado de paralelización del algoritmo y la gran cantidad y de parámetros y constantes tuvo como consecuencia realizar varias veces la operación de copiar del GPU al CPU y viceversa lo cual consume gran cantidad de tiempo.

4. Conclusiones y trabajo a futuro

En la paralelización del caso presentado, se siguieron los pasos para lograr paralelizar de manera óptima el algoritmo, sin embargo se concluye que el nivel de paralelización depende en gran medida de la arquitectura del GPU. En este caso se tuvo que reducir el grado de paralelización, es decir, la cantidad de instrucciones a paralelizar debido a la gran limitación de memoria y de ncleos del GPU disponibles. Por otro lado, es indispensable diseñar óptimamente el algoritmo paralelizado de acuerdo con las características técnicas del GPU. Como trabajo a futuro se plantea la posibilidad de experimentar con otras arquitecturas más potentes y con mayores recursos. Asimismo, es necesario calcular con mayor precisión el grado de paralelización.

Referencias

1. Quintão Pereira, F. Magno, Beatriz Perez, N., Marcelo Berón, M.: Algoritmo paralelo basado en el modelo filter - stream y la infraestructura watershed para búsquedas por similitud en espacios métricos.

2. García Alonso J. M., Berrocal Olmeda J. J., Murillo Rodríguez J. M.: ParallelJ: Entorno de desarrollo y simulación de programas.
3. Aracena Pizarro D., Danerí Alvarado N.: Detección de puntos clave mediante SIFT paralelizado en GPU. Revista chilena de ingeniería, 21(3), pp. 438–447 (2013)
4. García Armenteros A., Luque Polo G., Alba Torres E.: Influencia de Métricas Paralelas en el Análisis de Algoritmos Paralelos Metaheurísticos. Serie Científica de la Universidad de las Ciencias Informáticas, 4(4) (2011)
5. Uribe Paredes R., Cazorla D., Arias E., Sánchez J.L.: Un sistema heterogneo Multicore/GPU para acelerar la búsqueda por similitud en estructuras métricas. Revista chilena de ingeniería, 22(1), pp. 26–40 (2014)
6. Hidrobo F. J., Aguilar J.L.: Sistema Manejador de Ambientes Reconfigurables para procesamiento paralelo/Distribuido. Computación y Sistemas, 3(3), pp. 169–181 (2000)
7. NVIDIA CUDA C BEST PRACTICES GUIDE DG –05603–001 v5.0 (May 2012)
8. Dally, W.J.: Análisis de la ejecución de k-Aria n-Cube redes de interconexión. IEEE Transactions on Computadoras, 39(6), pp. 775–785 (1990)
9. Garcia, J.M. Duato, J.: Dynamic reconfiguration of multicomputer networks: limitations and tradeoffs. In: Proceedings of Euromicro Workshop on Parallel and Distributed Processing, pp. 317–323 (1993)
10. Flynn, M.: Some Computer Organizations and Their Effectiveness. IEEE Transactions on Computers, vol.C-21(9), pp. 948–960 (1972)
11. Bauch, A., Braam, R., Maehle, E.: DAMP - Un sistema multiprocesador reconfigurable dinámica con una red de conmutación distribuida. Spriger, vol. 487, pp. 495–504 (1991)
12. Serbediija, N.B.: Simulating Artificial Neural Networks on Parallel Architectures (1996)

Análisis de datos de calidad del aire de la Zona Metropolitana del Valle de México mediante técnicas de agrupamiento

Pablo Camarillo-Ramírez, Abraham Sánchez-López, Luis J. Calva-Rosales, Ivan Peréz-Vázquez

Facultad de Ciencias de la Computación,
Benemérita Universidad Autónoma de Puebla, México

{pablo.camarillo, asanchez, luis.calva}@cs.buap.mx, navip_1301@hotmail.com

Resumen. El cambio climático representa uno de los problemas a los que la sociedad actual se enfrenta, debido a sus impactos en la salud de los seres vivos. Es por ello que las autoridades necesitan de herramientas que les brinden la información necesaria para tomar decisiones que disminuyan el impacto de dicho cambio. En este trabajo, se propone una estrategia para agrupar los datos de la calidad del aire de la zona metropolitana de la Ciudad de México de los años 1999 al 2002, para reconocer los patrones de mediciones en los contaminantes del ambiente que detonan en precontingencias y contingencias ambientales en la zona de la Ciudad de México.

Palabras clave: Calidad del aire, clustering, K-means.

1. Introducción

El ser humano a lo largo de su existencia ha ido cambiando su entorno para vivir de manera cómoda y segura, prueba de ello son los grandes alcances para transportarse por cielo, mar, tierra y el espacio. Los avances tecnológicos han facilitado los hábitos cotidianos, los negocios, la fabricación de grandes cantidades de productos, etc. Sin embargo, estos avances han llevado a un deterioro ambiental que amenaza seriamente el desarrollo actual y futuro de las naciones [14], [9].

La contaminación del aire o contaminación atmosférica es un problema que produce cambios climáticos en todo el mundo y afecta a la salud de millones de personas. Es por esta razón que las herramientas tecnológicas que contribuyan en el estudio de esta contaminación son de vital importancia en la elaboración de políticas que erradiquen la contaminación o que mitiguen los efectos de la misma.

Actualmente, diversas organizaciones y gobiernos han implementado mecanismos de medición de contaminantes en el aire para conocer los índices de la

calidad del aire de las diferentes regiones del planeta. Los índices de calidad del aire (ICA) son número usados por agencias del gobierno para determinar la calidad del aire. En la ciudad de México y en la zona metropolitana del valle de México (ZMVM) la contaminación del aire es medida con el índice metropolitano de calidad del aire (IMECA). El IMECA es usado para mostrar el nivel de contaminación y el nivel de riesgo que representa a la salud humana en un tiempo determinado y así poder tomar medidas de protección.

En [14] la autora propone una aplicación de la Inteligencia de Negocios (Bussiness Intelligence) para analizar datos de cambio climático para la zona sur del valle de Puebla. Los resultados de los procesos de Inteligencia de Negocios aplicados a los datos de la calidad del aire de la zona sur del valle de Puebla, que presenta la autora, señalan una relación muy fuerte entre la calidad del aire y las variables climáticas, también muestran que la calidad del aire respecto de los niveles de concentración de los contaminantes atmosféricos, se encuentra determinada por la presencia de partículas menores a 10 micrómetros (PM_{10}) y químicos de ozono (O_3). En la ciudad de Puebla se cuenta con 3 estaciones de medición de contaminantes, pero no se cuenta con un modelo propio que brinde información sobre la calidad del aire. En este trabajo, buscamos obtener conclusiones similares a las obtenidas por la autora del trabajo previamente descrito, aunque el enfoque de nuestra estrategia solo toma en cuenta la información de la calidad del aire para la Zona Metropolitana de la Ciudad de México.

Otro trabajo relacionado con el análisis de datos de cambio climático, es el presentado por Reyes Salazar et al. [10], en el cual se propone un algoritmo genético para agrupar los datos de cambio climático de la Zona Metropolitana del Valle de México. En este trabajo, los autores crean los *patrones* a partir de las mediciones de diversas estaciones de la región estudiada y las agrupan para determinar el tipo de contaminantes que son claves para la activación de una contingencia ambiental, según las normas de calidad del aire. Esta estrategia de agrupación resultó en la obtención de 10 grupos, en los que se pueden agrupar los datos de cambio climático, concluyendo que los patrones que representaban mayores niveles de medición en ciertos contaminantes, tales como (PM_{10}) y el ozono coinciden con mediciones altas de IMECAs.

En la estrategia propuesta en este trabajo, cada medición de contaminantes conocida es tomada como un *patrón* en el cual, los atributos del patrón, son los valores de cada contaminante y de esta forma su agrupamiento nos conducirá a conclusiones sobre la relación entre los valores de los contaminantes y la calidad del aire. La literatura ofrece diversas técnicas de agrupamiento de datos [9], sin embargo en nuestra estrategia hemos usado el método *K-means* para agrupar los datos de calidad del aire. En la sección 2 se presenta una breve reseña de los conceptos más importantes referentes al cambio climático y a las técnicas de agrupamiento, después, en la sección 3 se describe la estrategia propuesta, para que en la sección 4 se presenten e interpreten los resultados obtenidos y, finalmente 5 se presentan las conclusiones obtenidas y el trabajo futuro de esta propuesta.

2. Marco teórico

2.1. Cambio climático

El cambio climático es uno de los procesos más relevante del deterioro ambiental mencionado anteriormente. Para entender qué es el cambió climático, debemos mencionar que la Convención Marco de las Naciones Unidas sobre el Cambio Climático (CMUCC) dice que *"cambio climático" se entiende un cambio de clima atribuido directa o indirectamente a la actividad humana que altera la composición de la atmósfera mundial y que se suma a la variabilidad natural del clima observada durante periodos de tiempo comparables.* El clima es una descripción estadística de las condiciones de tiempo y sus variaciones, incluyendo condiciones promedio y extremas. Los gases de efecto invernadero juegan un rol importante en el deterioro del clima y causan el cambio climático. Los gases de efecto invernadero incluyen vapor de agua, dióxiodo de carbono (CO_2), metano (CH_4), óxido nitroso (N_2O) y algunos gases industriales tales como clorofluorocarbonos (CFC_s). Estos gases actúan como una manta aislante, manteniendo la superficie de la tierra más caliente al no reflejar los rayos del sol al espacio, manteniendo el calor en la superficie de la tierra más caliente de lo que debería estar si estos gases no se presentaran en la atmósfera. Una vez liberados a la atmósfera, muchos de estos gases permanecerán ahí por un largo periodo de tiempo [6,8,13].

2.2. Calidad del aire

Se entiende por contaminación atmosférica a la presencia, en la atmósfera, de sustancias en una cantidad que implique molestias o riesgo para la salud de las personas y de los demás seres vivos. La calidad del aire es medida por la concentración de los contaminantes aéreos, es decir a mayor presencia en el aire, menos es la calidad y consecuentemente mayor es el impacto en la naturaleza y los seres vivos [7].

De acuerdo a las normas mexicanas de calidad del aire, para reportar la calidad del aire, el índice emplea cinco categorías [12]. La Tabla resume las categorías de calidad del aire, a partir de las mediciones de IMECA.

Tabla 1. Categorías de calidad del aire.

Buena	Regular	Mala	Muy mala	Extremadamente mala
0 - 50	51 - 100	101 - 150	151 - 200	201 - 500

2.3. Agrupamiento de datos

Cuando se trabaja con grandes cantidades de información, es necesario agruparla para que ésta sea más fácil de manejar. Agrupar esta información incrementa la eficiencia de su administración y su interpretación [15]. Existen dos

grupos de técnicas de agrupamiento: Agrupamiento jerárquico y agrupamiento no jerárquico.

Agrupamiento jerárquico Los métodos jerárquicos generan una sucesión de particiones, donde cada partición se obtiene uniendo o dividiendo grupos. Dentro de los métodos jerárquicos se distinguen dos tipos:

1. Métodos aglomerativos. En estos, los nuevo grupos se forman uniendo los grupos existentes. La ventaja de estos métodos es su rapidez.
2. Métodos divisivos. A diferencia de los métodos aglomerativos, en estos métodos, los nuevo grupos se forman dividiendo los grupos existentes. La ventaja de estos métodos es que parten de la información global que hay en los datos que además el proceso de división no tiene por que seguir hasta que cada elemento forme un grupo.

Sin embargo, de acuerdo a [1], estos métodos suelen ser muy lentos y en general aplicables solo para datos con pocos casos.

Agrupamiento no jerárquico El agrupamiento no jerárquicos los datos se dividen en k particiones o grupos donde cada partición representa un grupo. El funcionamiento básico de los métodos de agrupamiento no jerárquico son:

1. Seleccionar K centroides iniciales, siendo K el número de grupos deseados.
2. Asignar cada patrón al grupo que le sea más cercano.
3. Reasignar o relocalizar cada observación a uno de los K grupos de acuerdo con alguna regla de paro.
4. Termina si no hay reasignaciones de los puntos o si la reasignación satisface la regla de parada. En otro caso se regresa al paso dos.

Validación de clusters: silueta de los clusters La silueta de un cluster se refiere a un método de interpretación y validación del agrupamiento de datos. La técnica provee una representación gráfica de que tan bien está situado cada objeto en su cluster. Es un método propuesto por [11]. El ancho de la silueta (silhouette width) de la i-ésima observación es definida por: $sil_i = \frac{(b_i - a_i)}{max(a_i, b_i)}$ Donde, a_i denota la distancia promedio entre la observación i y todas las otras que están en el mismo cluster de i, y b_i denota la distancia promedio mínima de i a las observaciones que están en otros clusters.

Las observaciones con ancho de silueta grande están bien agrupadas mientras aquellas con ancho de silueta baja tienden a estar ubicada en medio de dos clusters. Para un número de clusters dado K, el ancho de silueta promedio de la configuración de conglomerados será simplemente el promedio de sil_i sobre todas las observaciones. Es decir, $\overline{s} = \frac{\sum_i sil_i}{n}$ Kaufman y Rousseeuw [3] sugirieron estimar el número óptimo de cluster K para el cual el ancho de silueta promedio sea la mayor posible.

Después de haber hecho una breve revisión de los conceptos básicos de nuestro trabajo, en la siguiente sección se presenta nuestro propuesta de análisis de datos.

3. Estrategia de agrupamiento propuesta

La estrategia de agrupamiento propuesta para los datos de cambio climático consiste en emplear el método K-means. Las instancias se forman a partir de la información de las mediciones de contaminantes de la Zona Metropolitana del Valle de México, cada instancia consiste en un vector que contiene las mediciones de seis contaminantes criterio de cada hora de cierto año (desde 1995 hasta 2002), resultando en un total de 8760 instancias por año. Estas mediciones se agrupan empleando la técnica mencionada anteriormente y se validan los grupos generados a partir de la silueta de los clusters resultantes.

3.1. Preparación de los datos

La primera fase de nuestra estrategia consiste en preparar los datos para su agrupamiento. Los datos originales consisten en un conjunto de hojas de cálculo que contienen las mediciones de diversas estaciones, sin embargo muchas de las estaciones reportan valores negativos, lo cual es imposible e indican un fallo en la estación, es por ello que en este estudio solo nos basamos en una estación, que es la que reporta menos valores incongruentes. La presencia de estos valores incorrectos se corrigió sustituyendo esos valores incongruentes por la media aritmética de los valores correctos.

3.2. Agrupamiento con K-means

El algoritmo empleado para agrupar los datos de calidad del aire, es el algoritmo K-means, que es uno de los algoritmos de agrupamiento no jerárquico más utilizado para encontrar grupos (*clusters*). Lo anterior, debido a su fácil implementación y a su rápida ejecución [2]. Este algoritmo se presentó desde los años sesentas [4] [5]. Parte un problema de m atributos, cada instancia se traslada a un espacio *m-dimensional*. El centroide del cluster describe cada cluster y es un punto en el espacio *m-dimensional* alrededor del cual cada instancia es agrupada. La distancia más usada de la instancia al centroide del cluster es la distancia Euclidiana. El algoritmo de K-means consiste en dos principales pasos:

1. El paso de **asignación** consiste en mover cada instancia a la clase más cercana.
2. El paso de **re-estimación** consiste en recaluclar los centorides de las clases a partir de las instancias asignadas a cada clase(cluster).

Se repiten los dos pasos del algoritmo hasta que el paso de re-estimación produce un cambio mínimo de los centroides de las clases.

Una vez corregidos los datos, se procede a construir, de tal forma que se pueda usar el algoritmo K-means para su agrupamiento. Las instancias quedan formadas por las mediciones de los contaminantes criterio. Contaminantes criterio, es un termino usado internacionalmente para describir contaminantes aéreos que han sido regulados y son usados como indicadores de la calidad del aire. Los

contaminantes criterio son: El Ozono (O_3), el dióxido de sulfuro (SO_2), dióxido de nitrógeno (NO_2), monóxido de carbono (CO) y las partículas suspendidas (PM). Con esa información, las instancias se forman como vectores como a continuación se describe:

$$X = (X_1, X_2, X_3, X_4, X_5, X_6)$$

Donde:

- X_1 corresponde a la medición del Monóxido de Carbono (CO)
- X_2 corresponde a la medición del Dióxido de Nitrógeno (NO_2)
- X_3 corresponde a la medición del Oxido de Nitrógeno (NOX)
- X_4 corresponde a la medición del Ozono (O_3)
- X_5 corresponde a la medición de partículas suspendidas menores a 10 micrómetros (PM_{10})
- X_6 corresponde a la medición del Dióxido de Sulfuro (SO_2)

Es necesario mencionar que se agrega la medición del óxido de nitrógeno (NOX), ya que el ozono es creado por reacciones químicas entre este compuesto.

4. Experimentos y resultados

Como se menciono anteriormente, se tienen 8760 instancias por año, y son 8 años sobre los cuales se hizo el análisis de datos reportado en este trabajo. Esta información fue obtenida de la página oficial de calidad del aire del gobierno del Distrito Federal (`http://www.calidadaire.df.gob.mx/calidadaire/`). Se aplicó el algoritmo de agrupamiento K-means a cada conjunto de datos anuales, y para verificar el número de clusters encontrados el óptimo, se empleo la técnica de validación con las siluetas de los clusters. El resultado de la aplicación del algoritmo K-means a los 8 conjuntos de datos que representan las mediciones anuales de los datos de cambio climático, arrojan la información reportada en la Tabla 2.

Tabla 2. Número óptimo de grupos para los datos de calidad del aire.

Año	Número óptimo de grupos
1995	9
1996	9
1997	6
1998	6
1999	6
2000	6
2001	6
2002	6

4.1. Validación con las siluetas de los clusters

El número óptimo de grupos fue calculado a partir de la información obtenida de la silueta de cada ejecución del método K-means con diferente número de grupos. En la Tabla 3 se presentan las pruebas que se hicieron con un diferente número de grupos y el error que arroja la silueta de cada uno de ellos, para los 8 años de mediciones que se están estudiando en este trabajo. Se remarcan los resultados que presentan menores errores en las siluetas de cada prueba, con ello se justifica el número óptimo de grupos para cada medición anual. Los resultados de las pruebas muestran agrupaciones con mínimo 6 grupos y máximo 11 grupos, ya que con un número de grupos menor que 6 y mayor que 11 el error se incrementa y por fines prácticos se decidió omitir esos resultados.

Tabla 3. Errores arrojados por la silueta de cada agrupación.

Número de grupos / Año	6	7	8	9	10	11
1995	0.5037	0.5050	0.5050	**0.4943**	0.4943	0.5292
1996	0.4292	0.3922	0.4190	**0.3883**	0.4180	0.3922
1997	**0.3841**	0.4585	0.4083	0.4379	0.4585	0.4319
1998	**0.4372**	0.4415	0.4612	0.4612	0.4612	0.4386
1999	**0.4155**	0.4603	0.4674	0.4793	0.5228	0.5249
2000	**0.4218**	0.7771	0.8226	0.8712	0.8705	0.8608
2001	**0.4520**	0.4628	0.5340	0.4628	0.4694	0.5340
2002	**0.3124**	0.3928	0.3517	0.3795	0.3131	0.3517

Una vez que se tienen agrupados los datos de la calidad del aire de cada año, desde 1995 hasta el 2002, la Figura 1, muestra las siluetas de estos clusters.

Con los grupos obtenidos por el método de K-means para cada conjunto de datos anuales, se ha relacionado la información de los contaminantes asociada a cada cluster de datos. En las Tablas 4–11 se concentra la información sobre los valores máximos y mínimos para cada contaminante criterio en cada grupo. Con esta información se puede observar una clara tendencia que la calidad del aire esta fuertemente influenciada por las partículas suspendidas menores a 10 micrómetros (PM_{10}).

5. Conclusiones

El interés por realizar la presente investigación es para contestar las siguientes preguntas: ¿Existe un patrón en los registros de cada año?, ¿Sólo un contaminante criterio se dispara por medición?, ¿Cuáles son los contaminantes que se disparan con mayor frecuencia?. Estas preguntas no pueden ser contestadas con solo tener el registro de la calidad del aire en un momento determinado, si no

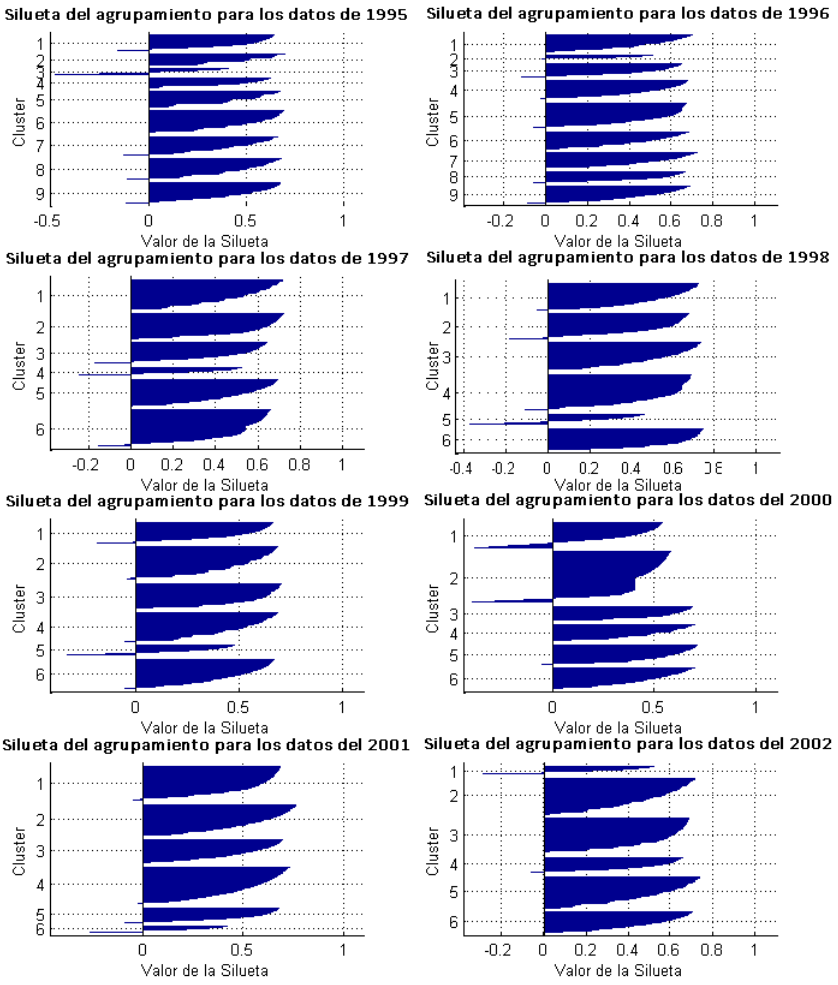


Fig. 1. Siluetas de los agrupamientos de los datos de calidad del aire anuales

Tabla 4. Valores máximos y mínimos para cada cluster I.

Datos del agrupamiento de los datos de calidad del aire de 1995							
Contaminante / Número de grupo		CO	NO_2	NOX	O_3	PM_{10}	SO_2
1	**Máximos**	15.7000	0.2390	0.4860	0.2690	134.0000	0.2400
	Mínimos	0.1000	0.0030	0.0040	0.0010	91.0000	0.0010
2	**Máximos**	12.0000	0.1950	0.4110	0.1950	24.0000	0.1730
	Mínimos	0.1000	0.0020	0.0030	0.0010	18.0000	0.0010
3	**Máximos**	12.4000	0.2870	0.4800	0.2920	782.0000	0.1280
	Mínimos	0.2000	0.0100	0.0050	0.0030	134.0000	0.0010
4	**Máximos**	10.6000	0.1540	0.3440	0.1630	17.0000	0.1620
	Mínimos	0.1000	0.0010	0.0010	0.0010	1.0000	0.0010
5	**Máximos**	7.3000	0.0840	0.2310	0.1720	32.0000	0.1450
	Mínimos	0.1000	0.0020	0.0010	0.0010	25.0000	0.0010
6	**Máximos**	13.8000	0.1480	0.4910	0.2460	67.0000	0.1760
	Mínimos	0.1000	0.0040	0.0030	0.0010	53.0000	0.0010
7	**Máximos**	8.7000	0.2360	0.3560	0.2070	41.0000	0.1950
	Mínimos	0.1000	0.0010	0.0020	0.0010	32.0000	0.0010
8	**Máximos**	13.3000	0.1310	0.4340	0.2090	52.0000	0.1190
	Mínimos	0.1000	0.0050	0.0020	0.0010	41.0000	0.0010
9	**Máximos**	13.1000	0.1730	0.4110	0.2350	91.0000	0.1300
	Mínimos	0.1000	0.0020	0.0030	0.0010	68.0000	0.0010

Tabla 5. Valores máximos y mínimos para cada cluster I.

Datos del agrupamiento de los datos de calidad del aire de 1996							
Contaminante / Número de grupo		CO	NO_2	NOX	O_3	PM_{10}	SO_2
1	**Máximos**	10.1000	0.1390	0.4210	0.1780	42.0000	0.1970
	Mínimos	0.1000	0.0040	0.0020	0.0010	30.0000	0.0010
2	**Máximos**	20.0000	0.2460	0.4250	0.2040	863.0000	0.1560
	Mínimos	0.3000	0.0110	0.0120	0.0040	239.0000	0.0010
3	**Máximos**	19.3000	0.3010	0.5000	0.2270	239.0000	0.1870
	Mínimos	0.2000	0.0090	0.0050	0.0020	151.0000	0.0010
4	**Máximos**	24.4000	0.2620	0.4920	0.2390	151.0000	0.1690
	Mínimos	0.1000	0.0090	0.0040	0.0010	104.0000	0.0010
5	**Máximos**	21.8000	0.2270	0.4930	0.2620	103.0000	0.1350
	Mínimos	0.1000	0.0030	0.0020	0.0010	75.0000	0.0010
6	**Máximos**	12.5000	0.1440	0.4440	0.2200	57.0000	0.1420
	Mínimos	0.1000	0.0050	0.0020	0.0010	42.0000	0.0010
7	**Máximos**	7.3000	0.1070	0.4050	0.2210	29.0000	0.1410
	Mínimos	0.1000	0.0050	0.0040	0.0020	18.0000	0.0010
8	**Máximos**	8.3000	0.0740	0.2340	0.1730	18.0000	0.1380
	Mínimos	0.1000	0.0020	0.0010	0.0020	1.0000	0.0010
9	**Máximos**	11.7000	0.1650	0.4880	0.2440	74.0000	0.1500
	Mínimos	0.1000	0.0010	0.0010	0.0010	57.0000	0.0010

Tabla 6. Valores máximos y mínimos para cada cluster I.

Datos del agrupamiento de los datos de calidad del aire de 1997							
Contaminante / Número de grupo		CO	NO_2	NOX	O_3	PM_{10}	SO_2
1	**Máximos**	10.6000	0.1170	0.2110	0.2250	37.0000	0.1610
	Mínimos	0.1000	0.0010	0.0030	0.0010	21.0000	0.0010
2	**Máximos**	10.9000	0.0910	0.2890	0.2090	21.0000	0.1420
	Mínimos	0.1000	0.0010	0.0010	0.0020	1.0000	0.0010
3	**Máximos**	12.8000	0.1540	0.4690	0.2920	168.0000	0.2260
	Mínimos	0.2000	0.0030	0.0130	0.0010	93.0000	0.0010
4	**Máximos**	16.9000	0.1960	0.4960	0.2130	912.0000	0.1430
	Mínimos	0.1000	0.0040	0.0180	0.0020	169.0000	0.0020
5	**Máximos**	8.9000	0.1210	0.2840	0.2600	57.0000	0.1630
	Mínimos	0.1000	0.0010	0.0030	0.0010	38.0000	0.0010
6	**Máximos**	12.7000	0.1490	0.3790	0.2710	92.0000	0.2300
	Mínimos	0.1000	0.0010	0.0020	0.0010	57.0000	0.0010

Tabla 7. Valores máximos y mínimos para cada cluster II.

Datos del agrupamiento de los datos de calidad del aire de 1998							
Contaminante / Número de grupo		CO	NO_2	NOX	O_3	PM_{10}	SO_2
1	**Máximos**	11.3000	0.1980	0.4990	0.2500	108.0000	0.2380
	Mínimos	0.1000	0.0020	0.0030	0.0030	67.0000	0.0020
2	**Máximos**	8.2000	0.1170	0.4350	0.1960	43.0000	0.1580
	Mínimos	0.1000	0.0030	0.0010	0.0030	28.0000	0.0010
3	**Máximos**	8.1000	0.1800	0.2530	0.1490	16.0000	0.0940
	Mínimos	0.1000	0.0030	0.0010	0.0020	0.0000	0.0010
4	**Máximos**	9.6000	0.1230	0.3540	0.1650	27.0000	0.1880
	Mínimos	0.1000	0.0030	0.0010	0.0030	16.0000	0.0010
5	**Máximos**	12.9000	0.2160	0.4960	0.2230	686.0000	0.1530
	Mínimos	0.2000	0.0040	0.0040	0.0050	109.0000	0.0020
6	**Máximos**	9.9000	0.1920	0.4320	0.2160	67.0000	0.1990
	Mínimos	0.1000	0.0040	0.0010	0.0030	43.0000	0.0010

Tabla 8. Valores máximos y mínimos para cada cluster II.

Datos del agrupamiento de los datos de calidad del aire de 1999							
Contaminante / Número de grupo		CO	NO_2	NOX	O_3	PM_{10}	SO_2
1	**Máximos**	11.3000	0.1980	0.4990	0.2500	108.0000	0.2380
	Mínimos	0.1000	0.0020	0.0030	0.0030	67.0000	0.0020
2	**Máximos**	8.2000	0.1170	0.4350	0.1960	43.0000	0.1580
	Mínimos	0.1000	0.0030	0.0010	0.0030	28.0000	0.0010
3	**Máximos**	8.1000	0.1800	0.2530	0.1490	16.0000	0.0940
	Mínimos	0.1000	0.0030	0.0010	0.0020	0.0000	0.0010
4	**Máximos**	9.6000	0.1230	0.3540	0.1650	27.0000	0.1880
	Mínimos	0.1000	0.0030	0.0010	0.0030	16.0000	0.0010
5	**Máximos**	12.9000	0.2160	0.4960	0.2230	686.0000	0.1530
	Mínimos	0.2000	0.0040	0.0040	0.0050	109.0000	0.0020
6	**Máximos**	9.9000	0.1920	0.4320	0.2160	67.0000	0.1990
	Mínimos	0.1000	0.0040	0.0010	0.0030	43.0000	0.0010

Tabla 9. Valores máximos y mínimos para cada cluster II.

Datos del agrupamiento de los datos de calidad del aire del 2000							
Contaminante / Número de grupo		CO	NO_2	NOX	O_3	PM_{10}	SO_2
1	**Máximos**	13.4000	0.2260	0.4910	0.2100	623.0000	0.4110
	Mínimos	0.4000	0.0040	0.0020	0.0020	78.0000	0.0030
2	**Máximos**	10.5000	0.1780	0.4310	0.2200	78.0000	0.3890
	Mínimos	0.1000	0.0030	0.0030	0.0010	46.0000	0.0010
3	**Máximos**	5.6000	0.0720	0.1520	0.1560	13.0000	0.1510
	Mínimos	0.1000	0.0010	0.0040	0.0030	0.0000	0.0010
4	**Máximos**	5.9000	0.0810	0.2390	0.1720	23.0000	0.1680
	Mínimos	0.1000	0.0030	0.0040	0.0020	13.0000	0.0010
5	**Máximos**	8.7000	0.1230	0.3760	0.1910	46.0000	0.2260
	Mínimos	0.2000	0.0030	0.0010	0.0020	34.0000	0.0010
6	**Máximos**	8.1000	0.0990	0.3460	0.1800	34.0000	0.3380
	Mínimos	0.1000	0.0020	0.0040	0.0010	23.0000	0.0010

Tabla 10. Valores máximos y mínimos para cada cluster II.

Datos del agrupamiento de los datos de calidad del aire del 2001							
Contaminante / Número de grupo		CO	NO_2	NOX	O_3	PM_{10}	SO_2
1	**Máximos**	8.5000	0.0520	0.1750	0.1370	32.0000	0.3290
	Mínimos	0.1000	0.0040	0.0030	0.0010	1.0000	0.0010
2	**Máximos**	11.0000	0.0820	0.3210	0.1670	73.0000	0.4930
	Mínimos	0.1000	0.0040	0.0050	0.0010	53.0000	0.0010
3	**Máximos**	13.6000	0.0970	0.4020	0.1890	100.0000	0.4980
	Mínimos	0.6000	0.0060	0.0060	0.0020	74.0000	0.0010
4	**Máximos**	7.3000	0.0820	0.1860	0.1750	53.0000	0.4920
	Mínimos	0.1000	0.0040	0.0030	0.0010	33.0000	0.0010
5	**Máximos**	13.9000	0.1130	0.4610	0.1960	141.0000	0.4980
	Mínimos	0.9000	0.0070	0.0070	0.0020	100.0000	0.0030
6	**Máximos**	11.6000	0.1600	0.4130	0.1880	721.0000	0.4690
	Mínimos	1.2000	0.0100	0.0150	0.0020	142.0000	0.0040

Tabla 11. Valores máximos y mínimos para cada cluster III.

Datos del agrupamiento de los datos de calidad del aire del 2002							
Contaminante / Número de grupo		CO	NO_2	NOX	O_3	PM_{10}	SO_2
1	**Máximos**	11.9000	0.1640	0.3930	0.1680	368.0000	0.2410
	Mínimos	0.1000	0.0050	0.0120	0.0020	97.0000	0.0020
2	**Máximos**	7.7000	0.1230	0.3620	0.1590	36.0000	0.1880
	Mínimos	0.1000	0.0050	0.0100	0.0010	22.0000	0.0010
3	**Máximos**	6.5000	0.1340	0.2980	0.1450	21.0000	0.2240
	Mínimos	0.1000	0.0040	0.0050	0.0010	0.0000	0.0010
4	**Máximos**	11.3000	0.1470	0.3460	0.1980	97.0000	0.2680
	Mínimos	0.2000	0.0070	0.0140	0.0010	69.0000	0.0010
5	**Máximos**	6.9000	0.1320	0.3110	0.1640	51.0000	0.2230
	Mínimos	0.1000	0.0020	0.0050	0.0010	37.0000	0.0010
6	**Máximos**	8.0000	0.1180	0.3290	0.1760	69.0000	0.1810
	Mínimos	0.1000	0.0060	0.0120	0.0010	52.0000	0.0010

que se requiere del análisis de las mediciones de calidad del aire para observar cual es el comportamiento de los datos. Y obtener las conclusiones pertinentes.

La estrategia de agrupamiento presentada en este trabajo brinda una herramienta para el análisis de datos de calidad del aire, en específico se realizaron las pruebas con la información de la calidad del aire de la Zona Metropolitana del Valle de México. El estudio realizado ha demostrado que existe una variación importante en los datos de calidad del aire de un año a otro,, ya que el número de grupos varía de un año a otro, lo que con ayuda de expertos, se puede interpretar como una causa del cambio climático.

La interpretación exacta del agrupamiento obtenido en nuestro estudio no es una tarea trivial, debido a la falta de un modelo que identifique los contaminantes criterio que influyan en el aumento de los niveles de IMECAs y por consecuencia, en la declaración de una contingencia ambiental. Nuestra estrategia propone una forma de agrupar estos valores y puede ser empleada para cualquier otra región que tenga estaciones que midan los contaminantes criterio.

Referencias

1. Araujo, B.: Aprendizaje automático: conceptos básicos y avanzados : aspectos prácticos utilizando el software Weka. Pearson Prentice Hall (2006)
2. Davidson, I.: Understanding k-means non-hierarchical clustering. Tech. Rep. 02-2, State University of New York, 1400 Washington Ave., Albany 12205 (February 2002)
3. Kaufman, L., Rousseeuw, P.J.: Finding groups in data: an introduction to cluster analysis. John Wiley and Sons, New York (1990)
4. Max, J.: Quantizing for minimum distortion. Information Theory, IRE Transactions on 6(1), 7–12 (March 1960)
5. McQueen, J.: Some methods for classifications and analysis of multiattribute instances. In: Proceedings of the Fifty Berkley Symposium on Mathematics, Statics and Probability Vol. 1. pp. 284–269 (1967)
6. ONU: Convención marco de las naciones unidas (1992)
7. Osorio, M.A., Torrijos, T., Sánchez, A., Arroyo, O.: Preliminary analysis for an air quality management dss in the metropolitan valley of puebla, mexico. In: Proceedings of the 13th WSEAS International Conference on Mathematical Methods, Computational Techniques and Intelligent Systems, and 10th WSEAS International Conference on Non-linear Analysis, Non-linear Systems and Chaos, and 7th WSEAS International Conference on Dynamical Systems and Control, and 11th WSEAS International Conference on Wavelet Analysis and Multirate Systems: Recent Researches in Computational Techniques, Non-linear Systems and Control. pp. 210–215. MAMECTIS/NOLASC/CONTROL/WAMUS'11, World Scientific and Engineering Academy and Society (WSEAS), Stevens Point, Wisconsin, USA (2011)
8. Pachauri, R., Reisinger, A.: Contribución de los grupos de trabajo i, ii y iii al cuarto informe de evaluación del grupo intergubernamental de expertos sobre el cambio climático. Cambio climático 2007 p. 104 (2007)
9. Reyes-Salazar, J.E.: Descubriendo conocimiento en bases de datos de cambio climático con técnicas de cómputo suave. Master's thesis, Benemérita Universidad Autónoma de Puebla, Puebla, Mexico (2012)

10. Reyes-Salazar, J.E., Sanchez, A.: Analysis of air quality data in mexico city with clustering techniques based on genetic algorithms. In: Electronics, Communications and Computing (CONIELECOMP), 2013 International Conference on. pp. 27–31 (March 2013)
11. Rousseeuw, P.: Silhouettes: A graphical aid to the interpretation and validation of cluster analysis. J. Comput. Appl. Math. 20(1), 53–65 (Nov 1987)
12. Secretaria del Medio Ambiente: Imeca (2011), disponible en: `http://www.calidadaire.df.gob.mx/calidadaire/index.php?opcion=2&opcioninfoproductos=22`
13. Semarnat: Estaregia de méxico ante el cambio climático global (2008)
14. Torrijos-Muñoz, M.T.: Aplicación de la tecnología de inteligencia de negocios como una estrategia de análisis que contribuya a llevar un control de la calidad del aire y de las variables climáticas, así como a proporcionar instrumentos para mitigar el cambio climático en la zona sur del valle de Puebla. Ph.D. thesis, Universidad Popular Autónoma del Estado de Puebla, Puebla, Mexico (2012)
15. Ward, J.H.: Hierarchical grouping to optimize an objective function. Journal of the American Statistical Association 58(301), 236–244 (1963)

Comunicación entre agentes inteligentes mediante cálculo de eventos usando Erlang

José Yedid Aguilar López, Felipe Trujillo-Romero, Manuel Hernández Gutiérrez

Universidad Tecnológica de la Mixteca,
Oaxaca, México

jyedid.al@gmail.com, ftrujillo@mixteco.utm.mx, hg.manuel@gmail.com

Resumen. En este trabajo se presenta una nueva representación para la comunicación entre robots durante la realización de una tarea en específico. Estos robots también son llamados agentes inteligentes. Dichos agentes se deben de comunicar para llevar a cabo la tarea asignada de manera eficiente. Esta comunicación se planea de inicio en forma directa mediante el cálculo de eventos y se traduce al lenguaje Erlang para que los mensajes puedan ser enviados y recibidos por los agentes. Así mismo esta arquitectura define de manera implícita las acciones a ejecutar por los mismos. Para ello se define la de un enfoque denominado Modelación de Procesos Erlang. Este enfoque es validado en una tarea de búsqueda realizada por tres robots, los cuales deberán encontrar varios patrones en la región de búsqueda y transmitir los mensajes a sus compañeros. Los mensajes serán gestionados por un servidor central que fungirá como pizarra en la cual los agentes podrán escribir y/o leer dichos mensajes para determinar las acciones a realizar.

Palabras clave: Cálculo de eventos, Erlang, robótica distribuida, agentes inteligentes, modelado de procesos.

1. Introducción

Los agentes son procesos de software o robots que se mueven alrededor de un sistema para realizar tareas específicas ya que están especialmente programados para ello. El nombre de "agente" se deriva del hecho de que trabajan en nombre de un objetivo más amplio. Los agentes recopilan y procesan información, así como también pueden tener la capacidad de intercambiar esa información con otros agentes. A menudo, los agentes cooperan tal como una colonia de hormigas, pero también pueden tener competencia amistosa como en una economía de libre mercado. Entre los desafíos que se pueden presentar en los sistemas de agentes distribuidos son los mecanismos de coordinación entre los agentes, el control de la movilidad de los agentes, así como su diseño de software y las interfaces. Cabe señalar que la investigación y trabajo en agentes es interdisciplinario abarcando: inteligencia artificial, cómputo móvil, cómputo distribuido e ingeniería de software.

Dentro de un sistema distribuido nos interesa que los agentes desarrollen una tarea en particular pero que cooperen entre ellos para llevarla a cabo de una manera más rápida y eficiente. Ahora bien para entender todo esto es necesario partir de la definición de un sistema distribuido. Para ello usemos la definición dada por Lamport [7]: "Un sistema distribuido consiste de una colección de distintos procesos los cuales están espacialmente separados y se comunican unos con otros a través del intercambio de mensajes."

Como se puede observar en la definición anterior el paso de mensajes es fundamental para que los agentes puedan tener una buena comunicación entre ellos y por lo tanto llevar a cabo la tarea encomendada. Por lo tanto lo que se desea proponer es una alternativa de comunicación de los agentes. Este nuevo enfoque hará uso del paradigma paso de mensajes definiendo de inicio los posibles eventos mediante una notación de Modelación de Procesos Erlang, basada en el formalismo lógico cálculo de eventos y que tiene correspondencia unidireccional con la programación concurrente de Erlang. Esto significa que la implementación de los agentes serán tratados como procesos de este lenguaje. El primer paso sería hacer el modelado de la comunicación entre agentes a través de dicha notación. El segundo sería conseguir la programación de los agentes pasando el modelado a código Erlang.

Si esto se traslada o se lleva al campo experimental empleando robots que puedan cooperar para reconocer patrones en un entorno cerrado o para recuperar el mapa de dicho entorno puede ser útil. Lo anterior debido a que esta implementación en Erlang sería un módulo que poseería dicho robot para compartir la información de su recorrido lo que le permitiría indicar su estatus actual, auxiliar a otros a terminar su actividad o solicitar apoyo.

En este mecanismo de coordinación se asume que existirá un servidor central que se encargará de gestionar los mensajes el cual será montado sobre una plataforma fija. Mientras que los agentes o robots serán los clientes de dicho servidor.

Ahora bien, para comenzar a modelar el paso de mensajes se empleará la notación de Modelación de Procesos. Esta notación es una especificación de sistemas concurrentes. Con esta notación se contemplarán los procesos y tiempos de envíos y recepción de mensajes entre ellos. Dejando la estructura del mensaje y su contenido a resolver en la fase de implementación, aunque también es posible especificarla.

2. Antecedentes

En esta sección vamos a revisar algunos conceptos necesarios para el planteamiento y resolución del problema propuesto.

2.1. Calculo de eventos

El cálculo de eventos es un formalismo lógico basado en la narrativa para el razonamiento acerca de las acciones y sus efectos que puede ser útil para representar la ocurrencia de eventos del mundo real. Fue propuesto originalmente desde la perspectiva de programación lógica por Kowalski y Sergot [1] con la premisa de que

la lógica formal se puede utilizar para representar muchos tipos de conocimiento para muchos propósitos. Por ejemplo puede ser utilizado para la especificación formal de programas, bases de datos y el lenguaje natural en general. Esto debido a que para muchas de estas aplicaciones de la lógica es necesaria una representación del tiempo y de un tipo de lógica temporal no clásica [2].

El cálculo de eventos tiene acciones y propiedades que varían en el tiempo llamados eventos y fluentes respectivamente. Además, como se comentó, está basado en la narrativa la cual es una especificación de un conjunto de ocurrencias de eventos reales. Ello nos permite abordar ciertos fenómenos de forma más natural en el cálculo de eventos incluyendo: (1) eventos concurrentes, (2) tiempo continuo, (3) el cambio continuo, (4) eventos con duración, (5) los efectos no deterministas, (6) eventos parcialmente ordenados y (7) eventos desencadenados.

Desde la versión original del cálculo de eventos este ha evolucionado considerablemente y varios autores han propuesto sus versiones [3] [4] [5]. Sin embargo se puede denotar dos clases de cálculo de eventos principales: (1) el clásico y (2) el simplificado.

El cálculo de eventos clásico propuesto por Kowalski y Sergot en 1986 [1] trabaja con eventos que ocurren en un tiempo determinado, fluentes y períodos de tiempo. A partir de estos se definen los predicados que utiliza este cálculo de eventos.

Desarrollado por Kowalski [6] el cálculo de eventos simplificado a diferencia del cálculo de eventos clásico agrega el concepto de puntos de tiempo específicos en que un evento ocurre y quita el de período de tiempo en el que un evento pudo haber ocurrido.

Ambos enfoques del cálculo de eventos tienen sus predicados los cuales les permiten describir la ocurrencia de los eventos. Sin embargo y dado que el que se va a utilizar en este trabajo es el cálculo simplificado se presentarán los predicados de este. Por lo tanto, sea e = evento, f = fluente y t, t_1, t_2 = puntos de tiempo, se tienen los siguientes predicados:

Initially (f)	El fluente *f* es verdadero en el punto de tiempo 0.
HoldsAt (f, t)	El fluente *f* es verdadero en el tiempo *t*.
Happens (e, t)	El evento *e* ocurre en el tiempo *t*.
Initiates (e, f, t)	Si el evento *e* ocurre en el tiempo *t*, entonces el fluente *f* es verdadero después del tiempo *t*.
Terminates (e, f, t)	Si el evento *e* ocurre en el tiempo *t*, entonces el fluente *f* es falso después del tiempo *t*.
StoppedI n (t_1, f, t_2)	El fluente *f* es detenido entre los tiempos t_1 y t_2

2.2. Sistemas distribuidos

El término de sistemas distribuidos ha sido definido por varios autores [7, 8, 9, 10, 11]. Sin embargo, la definición en esencia no ha cambiado a lo largo de los años. En dichas definiciones, todos coinciden en que se trata de un conjunto de componentes que pudieran ser procesos o equipos de cómputo independientes en su funcionamiento

interno y que se comunican a través del envío de mensajes para tratar parte de la solución de un problema.

En los últimos años los sistemas distribuidos han tomado gran importancia debido a su presencia en la vida diaria, esto debido principalmente a la necesidad de [12, 10]: (a) Tener un entorno de cómputo distribuido geográficamente, (b) Acelerar el cálculo de cómputo, (d) Compartir recursos: hardware y software y (e) Tener la capacidad de recuperarse ante fallos del sistema.

Las características principales de los sistemas distribuidos son [11, 10]: (1) No poseer un reloj físico común, (2) No compartir la misma memoria, (3) Separación geográfica, (4) Concurrencia y (5) Autonomía y heterogeneidad.

Los sistemas distribuidos presentan a lo largo de su desarrollo retos de alta complejidad en las etapas de especificación, diseño, implementación y verificación, lo que ha dado lugar a que se elaboren propuestas de herramientas y formalismos en el caso de la especificación de sistemas distribuidos y con ello ayudar a realizar más eficientemente el resto del ciclo de desarrollo. Una de las ventajas de la especificación es que a partir de ella es posible desarrollar técnicas y herramientas que hagan que la etapa de implementación se realice de forma automática al menos parcialmente.

De esta manera el presente trabajo, coadyuvando al desarrollo de sistemas distribuidos con paso de mensajes, se centra en proponer una especificación para estos sistemas a través del cálculo de eventos, además de presentar una correspondencia entre está especificación y la sintaxis del lenguaje de programación Erlang, el cual aborda la complejidad del desarrollo a gran escala de sistemas concurrentes y distribuidos. Esta correspondencia que se menciona ayudará a convertir una notación del cálculo de eventos a código Erlang con las limitantes que más tarde se indicarán.

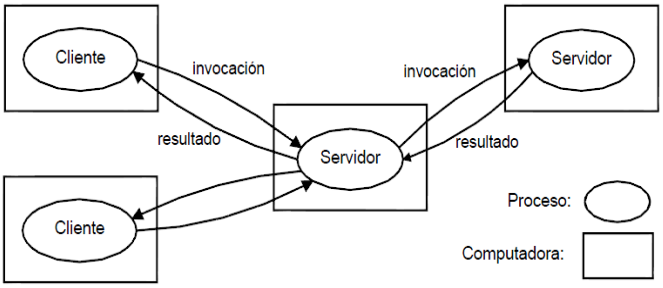


Fig. 1. Los procesos con roles Cliente o Servidor.

2.3. Arquitectura cliente/servidor

Si bien es cierto que existen varias arquitecturas empleadas en los sistemas distribuidos una de las más utilizadas es la arquitectura Cliente/Servidor [13, 14, 11] y es la que se va a discutir.

En la arquitectura Cliente/Servidor los procesos toman el rol de cliente o servidor. En particular, los procesos clientes interactúan con procesos servidores con el fin de

acceder a recursos que estos administran los cuales pueden estar hospedados en distintos equipos de cómputo, ver Fig. 1.

Los servidores pueden ser a su vez clientes de otros servidores, como lo muestra el gráfico. Por ejemplo, un servidor web es a menudo un cliente de un servidor de archivos local que gestiona los archivos en los que las páginas web se almacenan. Los servidores web así como la mayoría de los otros servicios de Internet son clientes del servicio DNS, el cual traduce los nombres de dominio de Internet en direcciones de red. Otro ejemplo relacionado con la web se refiere a los motores de búsqueda, los cuales permiten a los usuarios ver los resúmenes de la información disponible en las páginas web en los sitios de todo Internet.

La interacción entre cliente y servidor se da cuando el cliente hace una solicitud al servidor y entra en un tiempo de espera mientras este último trabaja para proporcionar la respuesta.

2.4. Programación en Erlang

El lenguaje de programación Erlang fue concebido para hacer frente a problemas relacionados con el desarrollo de sistemas distribuidos de tiempo real altamente concurrentes, buscando una solución para [15]: a) poder desarrollar este tipo de software de forma rápida y eficiente, b) disponer de sistemas tolerantes a fallos de software y hardware y c) poder actualizar el software sobre la marcha, sin detener la ejecución del sistema.

Las características que posee el lenguaje de programación Erlang y por las cuáles se creó son [16] [15]: (1) Construcciones de alto nivel, (2) Crear sistemas distribuidos, (3) Tolerante a fallos, (4) Escalabilidad, (5) Código en caliente, (6) Orientado a la Concurrencia y (7) el Paso de mensajes en lugar de memoria compartida.

Como Erlang fue diseñado pensando para tener múltiples tareas en ejecución simultáneamente resulta ser un soporte natural para la concurrencia. Este sistema concurrente utiliza el concepto de proceso para obtener una separación clara entre las tareas, lo que permite crear arquitecturas tolerantes a fallos y utilizar plenamente el hardware multinúcleo disponible hoy en día [16]. Por lo que el término proceso en el lenguaje de programación Erlang es un concepto fundamental debido a que es la unidad de concurrencia.

Un proceso representa una actividad en marcha. Esto es, la ejecución de una pieza de código de un programa y que es concurrente a otro proceso ejecutando también su propio código. La única manera en que los procesos interactúan es a través del paso de mensajes en donde el dato enviado va de un proceso a otro [15].

Los procesos en Erlang están separados unos de otros y asegurados para no perturbarse entre sí. Esto significa que los procesos son como un tipo de burbuja que proporciona aislamiento con otros procesos [16].

Cada proceso en Erlang tiene su propia memoria de trabajo y un buzón para recibir mensajes, mientras que los hilos en muchos otros lenguajes de programación y sistemas operativos son actividades concurrentes que comparten el mismo espacio de memoria.

Cuando un proceso necesita intercambiar información con otro entonces le envía un mensaje. Ese mensaje es una copia de sólo lectura de los datos que el emisor posee. La comunicación de procesos en Erlang siempre trabaja como si el receptor recibiera una copia del mensaje, incluso si el emisor se encuentra en la misma computadora. Esta distribución transparente de procesos permite a los programadores Erlang ver a la red como un conjunto de recursos. Una de las cosas que también Erlang hace, es que su máquina virtual balancea automáticamente la carga de trabajo sobre los procesadores disponibles. Por lo que los programas Erlang automáticamente se adaptan a distinto hardware.

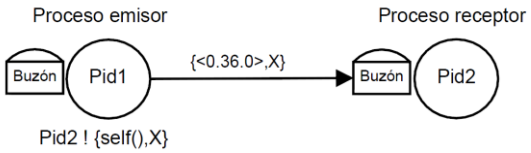


Fig. 2. Paso de mensajes entre procesos Erlang.

En Erlang las primitivas de paso de mensajes son asíncronas, a menudo el emisor no necesita saber si el mensaje llegó. Este método de comunicación asíncrono implica que el emisor no tiene que ser suspendido mientras se está entregando el mensaje, en particular si se envía el mensaje a través de un enlace de comunicación lento. El diagrama de la Fig. 2 muestra un ejemplo del envío de un mensaje de un proceso a otro.

3. Notación ProME

A la notación para la Modelación de Procesos Erlang se le denominó: ProME. La cual como ya se mencionó está basada en el formalismo lógico del cálculo de eventos. La notación ProME se centra en las acciones principales que se pueden hacer con los procesos de Erlang: crear procesos, enviar y recibir mensajes desde procesos.

En este trabajo se argumenta que la comunicación entre agentes puede realizarse vía paso de mensajes y modelar esto con la notación ProME, para más adelante conseguir las directrices de la implementación en Erlang dada la correspondencia de la notación con este último.

Antes, se mostrará un ejemplo de la modelación con esta notación describiendo la arquitectura *Cliente/Servidor* así como de su correspondiente representación en Erlang. Los puntos a considerar para realizar esto son:

1. Hay un único servidor y se llama *Server*,
2. Existe un cliente que se llama *Client* pero pueden haber varias instancias de este, y
3. La comunicación es síncrona entre el servidor y el cliente.

Sin embargo, también es necesario presentar los nombres de los eventos que utilizará la notación ProME del cálculo de eventos. Recordar que en la sección anterior se dijo que algunas de las acciones principales de la programación

concurrente en Erlang eran las de enviar y recibir mensajes. Por lo que estos eventos son:

1. *actuate* —involucrará la creación de una función Erlang.
2. *receive* —involucrará un patrón de la cláusula receive de Erlang.
   (a) *end* —es un caso especial del evento *receive*, para establecer cuándo un proceso debe terminar. Involucrará la definición de un patrón en la cláusula *receive* que haga que termine el proceso Erlang en cuestión.
3. *send* —involucrará el envío de un mensaje a un proceso con el operador de envío *!*.

Además se requiere que cada uno de estos eventos tenga una asignación lo cual significa que los eventos tendrán que estar relacionados con algunos valores. Por ejemplo, si se desea enviar un mensaje *Msg* a un proceso *Process1* entonces el evento *send* tendrá que conocer estos valores. La asignación quedaría como:

send(Process1,Msg)

Como se mencionó en secciones anteriores en una arquitectura *Cliente/Servidor* se puede ver a cada elemento como un proceso. Así que ahora vamos a describir el servidor como un proceso desde ambas perspectivas. Primero desde el punto de vista del cálculo de eventos y después realizando su correspondencia en lenguaje Erlang. Las funcionalidades serán las de lanzar el servidor, recibir peticiones, responder al cliente y como terminar la ejecución del servidor en caso de que se requiera.

3.1. Iniciar servidor

Para el evento *actuate*, aparte de relacionarse con algún valor se le tendrá que indicar qué es lo que tiene que accionar, valga la redundancia. En el caso del ejemplo *Server*, el nombre del servidor será el sufijo del nombre del evento *actuate* y quedará como:

actuateServer(Args)

Donde *Args* serían los argumentos que necesite el servidor expresados en los tipos de datos que emplea Erlang o en lenguaje natural. Con esta expresión lo que se tiene es que *actuateServer* es el evento que puede iniciar al proceso *Server* el cual necesita los argumentos *Args*. Empero hasta el momento todavía el servidor no inicia, sólo se está indicando qué iniciará y cómo se va a iniciar. Para poder iniciar el servidor se expresará con el predicado *Initiates* del cálculo de eventos, por lo que se tendría:

Initiates(actuateServer, liveServer, 0)

La sentencia anterior indica que se dará vida al servidor generando un fluente llamado *liveServer* con el evento *actuateServer* y es entonces cuando ya se pone en marcha.

Estas dos expresiones juntas representarían en Erlang las siguientes instrucciones:

```
1 -module(server).
2 -export([actuateServer/0, funtionServer/1]).
3
4 actuateServer()-> register(server,spawn(server,functionServer,[Args])).
5
6 functionServer(Args)-> functionServer(Args).
```

Este mapeo a código se lleva a cabo de la siguiente forma:

1. La línea 1 declara el uso de un módulo con nombre *server*. Este nombre se obtuvo del sufijo del evento *actuate*, que se formó como *actuateServer*.
2. La línea 2 declara las funciones *actuateServer* y *functionServer* que serán implementadas dentro del módulo. El nombre de las funciones se forma empleando también el sufijo *Server*. Los argumentos de la función *actuateServer* que lanza inicialmente al servidor, por default es cero, sin argumentos. En cambio los de la función *functionServer* será la cantidad de argumentos que se asignen en la expresión *actuateServer* del cálculo de eventos, en este caso se considerará *Args* como único argumento.
3. La línea 4 declara una función en la que se registra un proceso. El proceso que se registra es *serve*r, sufijo del evento *actuate*, y este proceso ejecutará la función *functionServer* del módulo *server*.
4. La línea 6 declara la función *functionServer* tomando como argumentos lo que se le haya asignado al evento *actuateServer*. El predicado *Initiates* hace que la función se invoque a sí misma, esto no es un problema para Erlang ya que soporta que los procesos puedan estar ejecutándose siempre.

3.2. Recepción en el servidor

Con el módulo *server* ya creado se definirán en este todas las funciones que necesite el servidor.com en el caso de: *actuateServer* y *functionSever*. Esta última será ejecutada por el proceso *server* y se le anexará las funcionalidades básicas del servidor como lo son las solicitudes que puede atender y las posibles respuestas para cada una de estas. Para el caso de las solicitudes que llegan, significa que se definirá una cláusula *receive* dentro de la función y se especificarán los patrones que tendrá. Con el evento *receive* de la notación ProME:

receive(From, Pattern1)

Para que este evento suceda se utiliza el predicado *Happens*:

Happens(receive, 1)

El segundo parámetro, valor 1, indica que debe realizarse el evento receive después del evento que sucede en el tiempo 0. La representación en código Erlang de esto consiste de un patrón tipo tupla con dos elementos. El primero de ellos es una variable no ligada que se encuentra a la espera de un Identificador de Proceso (PId) de algún proceso y el segundo es el patrón mismo que puede ser de cualquier tipo de dato Erlang. Dado que es la primera aparición del evento *receive* entonces se crea la constructiva *receive* de Erlang:

```
1 functionServer(Args)->
2     receive
3       {From,Pattern1}-> functionServer(Args)
4     end.
```

Para agregar otro patrón, la notación en el cálculo de eventos quedaría de la siguiente forma:

receive(From, Pattern2)
Happens(receive, 1)

Con las expresiones anteriores se genera una cláusula más en *receive* y se coloca como el segundo patrón lo cual hace que el código en Erlang se aumente una línea más.

3.3 Responder desde el servidor

Ahora bien, cada petición que llegue al servidor puede ser respondida por este. Para que el primer evento *receive* tenga su correspondiente evento *send*, después de las expresiones de recepción debería ir enseguida las siguientes:

send({ok,Response})
Happens(send, 2)

Lo cual haría que se modificara el cuerpo de la primera cláusula receive para agregar una operación de envío hacia el proceso del que se recibió la petición. La instrucción Erlang correspondiente a las expresiones anteriores se muestra en la línea 3 del siguiente código:

```
1 functionServer(Args)->
2     receive
3       {From,Pattern1}-> From ! {ok, Response},
4                         functionServer(Args);
5       {From,Pattern2}-> functionServer(Args)
6     end.
```

3.4. Terminar el servidor

Es recomendable que el servidor termine de buena manera haciendo antes una serie de acciones que se requieran sin que el proceso sea forzado a parar. Para eso se utiliza el evento *end* junto con el predicado *Terminates*. Esto queda expresado en el cálculo de eventos como:

end(Finish)
Terminates(end, liveServer, 1)

donde *Finish* será el patrón que tenga la sentencia *receive*, entonces cuando ocurre el evento *end* con el predicado *Terminates* hace que el fluente *liveServer* ahora sea falso. En código Erlang se agregara un patrón más en la función *functionServer*:

```
functionServer(Args)->
    receive
      {From,Pattern1}-> From ! {ok, Response},
                        functionServer(Args);
      {From,Pattern2}-> From ! {error, Reason},
                        functionServer(Args);
      {Finish}-> %to do some before to end the function
                 ''Process is finished.''
    end.
```

Estas instrucciones que se agregan a la función deberían de ser modificadas después para personalizar el fin del proceso. Incluso es posible continuar con eventos *send* para avisar a otros procesos de que el servidor terminará o enviar alguna información antes de esto.

Por razones de espacio se va a omitir el modelado del *Cliente*. Sin embargo se modela de forma análoga al *Servidor*.

4. Aplicación: búsqueda de patrones

Una vez mostrada la notación ProME pasaremos a validar dicha notación mediante la aplicación de la misma a una tarea de búsqueda. Esta tarea será realizada usando tres robots lo cual significa que se distribuirá el espacio de búsqueda para que esta se realice de una forma rápida y eficiente (Fig. 3).

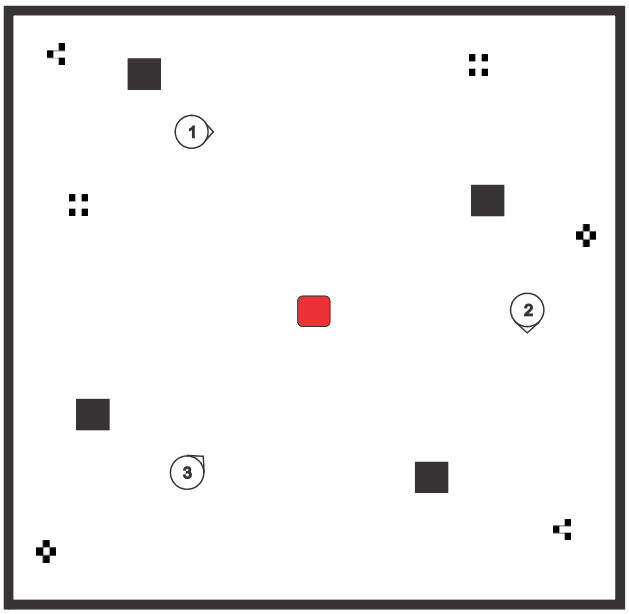


Fig. 3. Escenario de trabajo: los agentes son los círculos orientados, el cuadrado redondeado es el servidor y los cuadros negros son obstáculos. Además están los patrones a buscar.

4.1. Descripción del problema

En el escenario de trabajo los robots deberán buscar una serie de marcas en el piso e informar cada vez que han encontrado una de ellas en el instante que dos agentes encuentren la misma marca, en dos lugares diferentes, deberán de comunicarse para eliminarla. Es decir si la vuelven a encontrar no emitirán mensaje alguno. Además un mismo agente no podrá eliminar la marca aunque la encuentre en diferentes sitios del escenario. Cada marca es un patrón binario formado por una matriz de *3x3* (ver Fig. 4), tiene el tamaño necesario para que los sensores del robot puedan detectarlo al estar encima de la marca.

Dado que es una arquitectura Cliente/Servidor los agentes serán los clientes mientras que una computadora central fungirá como servidor. Este servidor será utilizado como pizarrón en donde se guardaran los diferentes mensajes emitidos por los agentes a los cuales pueden acceder los demás robots si poseen el patrón correcto.

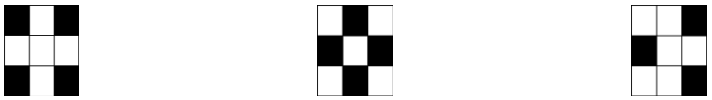


Fig. 4. Patrones utilizados.

4.2. Modelación con la notación ProME

Para el servidor *BBoard* las actividades que se modelarán son: permitir la sesión de un agente, enviar mensaje a otro usuario, enviar lista de agentes activos, terminar la sesión, enviar lista de patrones encontrados y determinar el fin de la actividad. La narrativa es la siguiente:

actuateBBoard()
Initiates(actuateBBoard, liveBBoard, 0)
receive(FromRobot, {login, RobotCredential})
Happens(receive, 1)
send(login accept)
Happens(send, 2)
send(login not accept)
Happens(send, 2)
receive(FromRobot,patterns)
Happens(receive, 1)
send({patterns, PatternsList})
Happens(send, 2)
receive(FromRobot, {invitation to, RobotID,Msg})
Happens(receive, 1)
send(RobotID, {invitation from, FromRobot,Msg})
Happens(send, 2)
receive(FromRobot,who)

Happens(receive, 1)
send({who, RobotList})
Happens(send, 2)
receive(FromRobot, logout)
Happens(receive, 1)

Los procesos del cliente se pueden dividir en dos partes: (1) Inicio de sesión y (2) actividades a realizar. Enseguida se muestra la primera narrativa para el cliente Robot donde se declara el inicio de sesión:

actuateRobotClient(RobotCredential)
Happens(actuateRobotClient, liveRobotClient, 1)
send(BBoard, {self(), login, RobotCredential})
Happens(send, 2)
receive(login_accept)
Happens(receive, 3)
end(login_not_accept)
Terminates(end, 3)

En caso de que el agente haya iniciado sesión exitosamente entonces recibirá como respuesta del servidor el átomo *login_accept* y ahora el agente ya puede realizar sus actividades, esto se modela en la siguiente narrativa:

actuateRobotClient()
Happens(actuateRobotClient, liveRobotClient, 4)
receive(Pattern_request)
Happens(receive, 5)
send(BBoard, {self(), patterns})
Happens(send, 6)
receive({invitation_to, RobotID,Msg})
Happens(receive, 5)
send(BBoard, {self(), invitation_to, RobotID,Msg})
Happens(send, 6)
receive({who, RobotList})
Happens(receive, 5)
end(logout_ request)
Terminates(end, 5)
send(BBoard, {self(), logout})
Happens(send, 6)

4.3 Implementation en Erlang

Se presenta aquí el código correspondiente a las tres narrativas de arriba descritas: dos del cliente *Robot* y una del servidor *BBoard*. Las dos narrativas del *Robot* tienen el mismo nombre de proceso, significa que se trata de la implementación de una

misma función pero que difiere en el tipo y en el número de argumentos. Comencemos por mostrar el código en Erlang de la narrativa correspondiente al inicialización del cliente, el cual queda como:

```
functionRobotClient(RobotCredential)->
  bboard ! {self(),login,RobotCredential},
  receive
    login_accept-> functionRobotClient();
    login_not_accept->
  end.
```

El código Erlang correspondiente al modelado de la actividad del cliente es:

```
functionRobotClient()->
  receive
    Pattern_request-> bboard ! {self(),patterns};
    {invitation_to,RobotID,Msg}-> bboard, {self(),invitation_to,RobotID,Msg};
    {who,RobotList}-> ;
    logout_request-> bboard ! {self(),logout};
  end.
```

Finalmente el código correspondiente al servidor *BBoard* es el siguiente:

```
-module(bboard).
-export([actuateBBoard/0, functionBBoard/0]).

actuateBBoard()-> register(bboard,spawn(bboard,functionBBoard,[])).

functionBBoard()->
  receive
    {FromRobot,login,RobotCredential}-> FromRobot ! login_accept,
                                        Fromrobot ! login_not_accept,
                                        functionBBoard();
    {FromRobot,patterns}-> FromRobot ! {patterns,PatternsList},
                           functionBBoard();
    {FromRobot,invitation_to,RobotID,Msg}-> RobotID ! {invitation_from,FromRobot,Msg},
                                            functionBBoard();
    {FromRobot,who}-> FromRobot ! {who,RobotList},
                      functionBBoard();
    {FromRobot,logout}->
  end.
```

Cabe mencionar que este ejemplo solo es una prueba de concepto y sirve para validar el modelado propuesto en una aplicación de robótica distribuida. En esta solo se modela el paso de mensajes que existirá entre los agentes al momento de llevar a cabo una tarea de esta índole. Por lo cual lo que se presenta es la evolución que se tiene en la comunicación realizada entre el servidor y los agentes durante la tarea asignada.

En esta tarea faltaría la codificación correspondiente a las acciones que realizaran cada uno de los agentes, las cuales solo se especifican. Sin embargo el código

generado se puede compilar en Erlang y es un ejemplo de que la metodología propuesta se puede aplicar a este tipo de sistemas distribuidos.

5. Conclusiones y perspectivas

En este trabajo se presentó un nuevo enfoque para realizar la comunicación de agentes inteligentes mediante el paso de mensajes. Este enfoque está basado en el cálculo de eventos el cual una vez definido se traduce al lenguaje Erlang para ser utilizado. Esto se realiza a partir de la notación ProME. El código generado en Erlang es solamente la comunicación entre el servidor y los clientes dejando la programación de las acciones al programador.

Para validar este enfoque se propuso una tarea de búsqueda de patrones usando tres agentes que se comunicaban para indicar cual patrón habían encontrado con la finalidad de descartarlos cuando dos robots encontraran el mismo patrón en dos sitios diferentes. De esta actividad se presentó la evolución de los eventos en el cálculo de eventos así como el código generado en Erlang.

Si bien no se tienen resultados prácticos

Entre los trabajos que se tienen considerados para continuar este proyecto se pueden mencionar principalmente: (1) Ampliación de la gramática de la notación ProME para que se puedan modelar diferentes actividades de propósito general, (2) Crear un software de traducción que permite la conversión a partir del cálculo de eventos hacia el lenguaje Erlang.(3) Validar el enfoque mediante la resolución de otro tipo de problemas y (4) Implementar la totalidad del problema de reconocimiento de patrones en simulación y en robots reales.

Referencias

1. R. Kowalski and M. Sergot: A logic-based calculus of events. New Gen. Comput., 4(1):67–95 (1986)
2. J. Mccarthy and P. J. Hayes: Some Philosophical Problems from the Standpoint of Artificial Intelligence. Machine Intelligence, pp. 463–502, Edinburgh University Press (1969)
3. R. Miller and M. Shanahan: Some alternative formulations of the event calculus. Computer Science; Computational Logic; Logic programming and Beyond, pp. 452–490, Springer Verlag (2002)
4. E. Mueller: Event calculus. In: F. van Harmelen, V, Lifschitz, B, Porter, editors, Handbook of Knowledge Representation, pp. 671–708. Elsevier (2008)
5. V. Alexiev: The event calculus as a linear logic program (1995)
6. R. Kowalski: Database Updates in the Event Calculus. Journal of Logic Programming, pp. 121–146 (1992)
7. L. Lamport: Time, clocks, and the ordering of events in a distributed system. Commun. ACM, 21(7):558–565 (1978)

8. M. Singhal and N. G. Shivaratri: Advanced Concepts in Operating Systems. McGraw-Hill, Inc. (1994)
9. A.S. Tanenbaum and M. van Steen: Distributed Systems: Principles and Paradigms. Pearson Prentice Hall, 2 edition (2007)
10. A.D. Kshemkalyani and M. Singhal: Distributed Computing: Principles, Algorithms, and Systems. Cambridge University Press (2008)
11. G.F. Coulouris, J. Dollimore, T. Kindberg, and G. Blair: Distributed Systems: Concepts and Design. Addison-Wesley, 5 edition (2011)
12. S. Ghosh: Distributed Systems: An Algorithmic Approach. Chapman & Hall/CRC Computer & Information Science Series. Taylor & Francis (2007)
13. T. Özsu and P. Valduriez: Principles of Distributed Database Systems. Springer (2011)
14. I. Sommerville: Software Engineering. Pearson Education, 9 edition (2011)
15. F. Cesarini and S. Thompson: Erlang Programming. O'Reilly Media (2009)
16. M. Logan, E. Merritt, and R. Carlsson: Erlang and OTP in Action. Manning (2011)

Una herramienta de propósito general para el plegado de proteínas con técnicas probabilísticas

Luis J. Calva-Rosales, Abraham Sánchez-López, Pablo Camarillo-Ramírez, Juan Carlos Conde-Ramirez

Facultad de Ciencias de la Computación,
Benemérita Universidad Autónoma de Puebla, México

{luis.calva, asanchez, pablo.camarillo, juan.conde}@cs.buap.mx

Resumen. La importancia de este trabajo radica en el hecho de que el estudio de plegado molecular tiene el fin de prevenir y entender mutaciones que ponen en peligro la vida de los seres vivos. El plegado molecular de proteínas es de vital importancia para entender los factores fisiológicos externos e internos que provocan que una proteína monomérica pase de un estado a otro. En este articulo se describe el desarrollo de una herramienta que calcula y simula el plegado de proteínas, utilizando técnicas de robótica como PRM (Probabilistic Roadmap Methods). Actualmente la cantidad de proteínas analizadas es escasa dado que las herramientas existentes son caras y necesitan de costosos equipos para realizar esta tarea. Por lo cual se desarrollo esta herramienta utilizando C++ con librerías QT como lenguaje de programación, junto con OpenGL y se consume el servicio Web DSSP para la asignación de estructuras secundarias en bancos de datos de proteínas.

Palabras clave: Plegado molecular, proteínas, PRM, simulación.

1. Introducción

Entre las muchas áreas de la bioinformática, existe la biología computacional estructural, la cual se encarga de estudiar las estructuras biológicas (ADN, Proteínas, etc) con la meta de obtener características, información, verificar comportamientos o calcular energía y poder estudiar sus efectos en los seres vivos [17].

En particular, las proteínas son estructuras fundamentales para todos los seres vivos. Cada proteína consiste de una secuencia de residuos de aminoácidos [5]. A su vez, cada aminoácido en una proteína es llamado *residuo* porque pierde dos átomos de hidrógeno y uno de oxigeno durante la formación de la *cadena péptida* entre dos aminoácidos adyacentes. De esta manera, bajo ciertas condiciones fisiológicas una sola proteína puede llegar a formar una estructura tridimensional compacta y estable; conocida como el estado nativo de una proteína [3]. El proceso para formar un estado nativo se llama plegado de proteínas. Existen dos problemas principales en el plegado de proteínas [16,13,14]:

1. **Predicción de las estructuras del estado nativo de la proteína**. Este problema es normalmente referenciado como la predicción de la estructura nativa de una proteína.
2. **Problema del plegado de la proteína**. Trata del estudio de la secuencia de las transiciones realizadas por los aminoácidos dinámicamente a partir de un estado no estructurado a la estructura nativa (única).

Este último se centra en el proceso de doblado dinámico y temas relacionados con la identificación de los caminos y el cálculo de la cinética de plegamiento.

Para el desarrollo de esta herramienta se utilizaron técnicas de robótica para modelar y configurar las proteínas con el fin de obtener energías y caminos que concuerden con características biológicas.

El contenido de este articulo esta distribuido como sigue. En la Sección 2 se detalla brevemente el estado del arte y se establece la relación con la planificación de movimientos en robótica. Posteriormente, la Sección 3 establece los elementos esenciales a considerar destacados por otros trabajos relacionados. Por su parte la Sección 4 describe el enfoque seguido durante la investigación para construir una herramienta que realiza el cómputo del plegado de proteínas en modo gráfico, así como las técnicas utilizadas. En la Sección 5 se definen los parámetros y las características de las pruebas realizadas con la herramienta propuesta, junto con los resultados medidos en tiempo y posibles caminos obtenidos. Al final, la Sección 6 muestra las conclusiones y el trabajo futuro de esta investigación.

2. Problema del plegado proteínico

Las proteínas y *polipéptidos* son compuestos de enlaces de aminoácidos, esas composiciones son conocidas como la estructura primaria de la proteína. Un aminoácido es una molécula orgánica simple que consiste de un *amino* básico (receptor de hidrógeno), unido a un ácido (donante de hidrógeno) a través de un único átomo de carbono intermedio. Cada aminoácido consiste de un átomo de carbono *tetrahédrico* central conocido como la *alpha* (α) de carbono (C^{α}) el cual tiene cuatro enlaces: un átomo de hidrógeno, un grupo *amino* receptor de átomo (NH_3^+), un grupo ácido pierde un átomo (COO^-), y una cadena lateral distintiva o grupo R que hace la diferencia entre los diferentes aminoácidos (Figura 1).

Entonces el proceso de plegado es importante por varias razones, ya que puede proporcionar la idea de cómo es que se pliegan las proteínas y puede ayudar a entender los factores que controlan este proceso en algunas proteínas (mecanismos). Esta área de investigación es de gran importancia práctica ya que existen algunas enfermedades devastadoras provocadas por plegamientos no nativos de la proteína, como la *encefalopatía espongiforme bovina*[1] [12], en estos casos es importante entender por qué ocurren estos plegados y cómo se podrían prevenir.

[1] La enfermedad de las vacas locas

2.1. Relación con la planificación de movimientos

La planificación de movimientos es un problema fundamental de la robótica que ha sido investigado por más de tres décadas. Se han propuesto una gran variedad de algoritmos para calcular los movimientos factibles de múltiples cuerpos y sistemas en diferentes espacios. En los últimos años, muchos de estos algoritmos han empezado a pasar las barreras de la robótica, encontrando aplicaciones en otros campos como la manufacturación industrial, la animación por computadora y la biología computacional estructural.

Debido a la gran similitud entre las proteínas y los robots manipuladores, en términos de movimiento, se pueden aplicar los algoritmos de planificación utilizados en la robótica en el campo de las proteínas. Esta visión es importante en el sentido de que se pueden obtener diversos caminos que pueden ayudar a entender el comportamiento de la proteína que se esta estudiando.

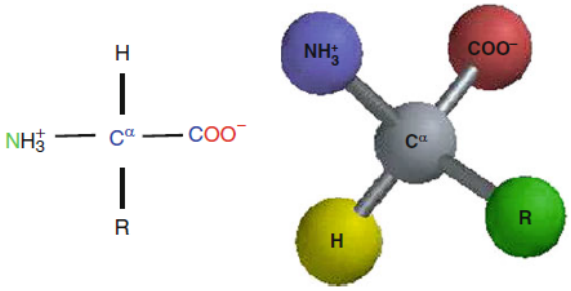


Fig. 1. Forma general para un aminoácido (izquierda), y disposición *tetrahédrica* espacial de un aminoácido (derecha).

3. Trabajo relacionado

Dentro del campo del diseño molecular[2] y de acuerdo a la literatura, existen diferentes maneras en las cuales se puede llegar a modelar una proteína, desde coordenadas cartesianas hasta complejos modelos físicos.

En [10] se muestran los tres diferentes grados de libertad internos de una proteína:

1. El **largo de enlace** que es la distancia de separación entre un par de átomos unidos el cual tiende a variar muy poco.
2. El **ángulo de enlace** que sólo tiene sentido cuando consideramos 3 átomos, y es el ángulo formado por los ejes imaginarios que unen el núcleo del átomo central con los núcleos de los átomos unidos a él.

[2] Ciencia que estudia la estructura y funcionamiento de estructuras moleculares a través de la construcción de modelos físicos o computacionales

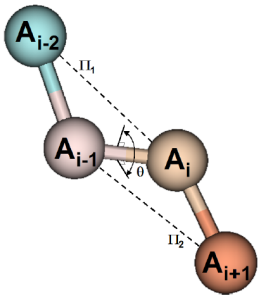


Fig. 2. Ángulo diedro definido como el ángulo más pequeño entre el plano π_1 y π_2.

3. El **ángulos diedros** que es definido por cuatro consecutivos enlaces de átomos como se muestra en la Figura 2.

En la Figura 3 se muestra los ángulos de libertad diedros del aminoácido *Arginina*, dependiendo el aminoácido los ángulos diedros en la cadena lateral pueden variar, pero los ángulos diedros de la cadena *péptida* son constantes(2) en todos los aminoácidos.

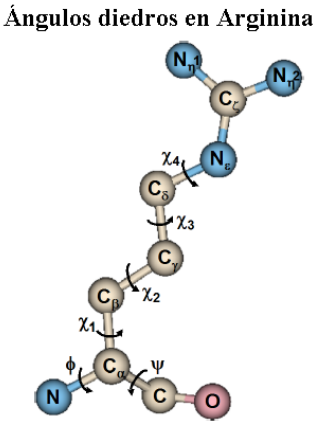


Fig. 3. Las cadenas laterales *diedras* se designan por X y un subíndice. Las cadenas internas *diedras* se designan por Φ y Ψ.

En este trabajo [20] la proteína es modelada como un robot articulado. Solo los ángulos diedros de la cadena *péptida* son utilizados como grados de libertad, los ángulos diedros de las cadenas laterales, el largo de enlace y el grado de enlace son considerados valores constantes. De esta manera una proteína modelada con nuestra aproximación contara con $2k$ grados de libertad.

Para la realización de un modelo como este se utiliza la convención *Denavit-Hartember* que permite modelar y configurar la estructura de una proteína. Se utilizó este enfoque y no uno de agrupación porque permite de una manejar todos los ángulos diedros internos de la proteína [20].

4. Herramienta propuesta

Para el desarrollo de esta herramienta se implementaron diversos módulos que a continuación se explicaran.

Para obtener la información necesaria para modelar una molécula se desarrollo un **interprete** para cargar archivos de tipo PDB que se pueden obtener de la página `www.rcsb.org` (*Protein Data Bank*), la descripción de estos archivos se puede encontrar en [7].

Una vez obtenida la información del archivo PDB que se desea estudiar, se desarrollo un **modulo de modelado y diseño molecular** el cual permite manipular y visualizar la proteína.

El modulo desarrollado es el **modulo que contiene la algoritmia** que permite emplear el algoritmo PRM para resolver el problema de plegado de proteína. En este caso, la Figura 4 permite observar el comportamiento del plegamiento realizado por la proteína.

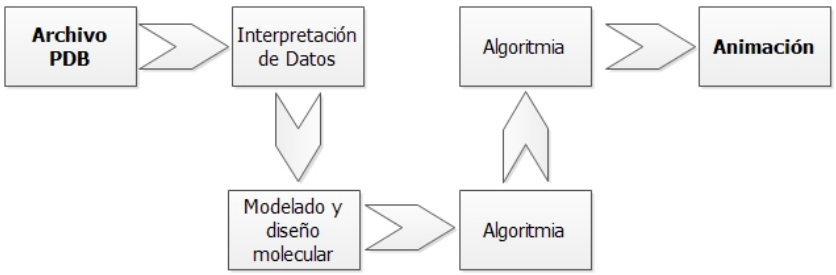


Fig. 4. Una visión general de la arquitectura propuesta del sistema.

4.1. Técnicas de robótica

Ya que la meta de los algoritmos de planificación de movimiento es ayudar a que un robot que se encuentra en un estado inicial encuentre un camino factible a configuración meta [15], los planificadores deben de ser capaces de garantizar una solución válida o determinar si ésta existe o no. La búsqueda de un camino requiere un tiempo exponencial que se puede incrementar dependiendo la cantidad de ángulos de libertad que tenga el robot. En el caso de las proteínas y en nuestro modelo la cantidad de grados de libertad está directamente relacionada al número de aminoácidos que esta contenga.

La herramienta desarrollada resuelve el problema del plegado de proteínas utilizando técnicas basadas en un mapa de ruta probabilístico (*probabilistic roadmap, PRM*) [9]. PRM es un algoritmo basado en muestras aleatorias de un espacio de configuraciones definido como $C - space$, donde si una muestra es factible es agregada a un grafo esta etapa se conoce como **etapa de muestreo**. Posteriormente se realiza la conexión de los mismos utilizando medidas de distancia y de restricción (en el caso de las proteínas restricciones energéticas) conocida como **etapa de conexión**. Una vez que existe un camino entre la configuración inicial y la configuración meta se procede a utilizar un algoritmo de búsqueda como lo es $A*$ o *Dijsktra*, a ésta se le denomina **etapa de consulta**. La meta de este algoritmo es encontrar un camino entre la configuración inicial y meta.

4.2. Consideraciones importantes

El problema de plegado de proteínas tiene diferencias notables de la aplicación usual de PRM. Las colisiones tradicionales son remplazadas por configuraciones con poca energía, en este caso se utilizó el algoritmo de BioCD para realizar esta tarea [4][19]. De esta manera se miden las pequeñas diferencias energéticas que se realizan en cada parte del proceso de PRM. En la aplicación general de PRM es suficiente encontrar un solo camino, en el caso del plegado de proteínas es importante la calidad del camino, se busca el camino más favorable.

En la Figura 5 podemos observar las diferentes etapas del desarrollo de PRM en el campo de las proteínas: (a) etapa de muestreo, (b) etapa de conexión y (c) etapa de consulta [1][2].

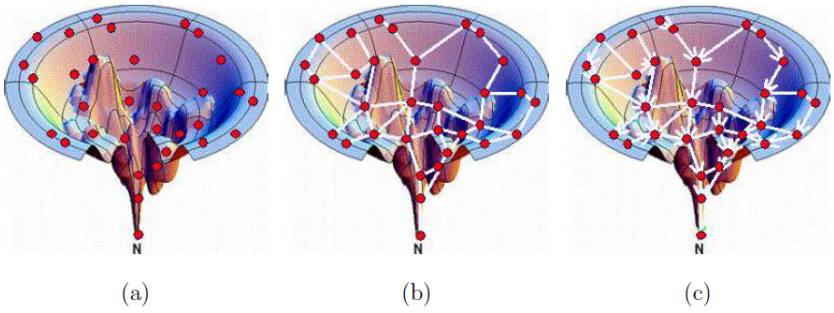


Fig. 5. Aplicación de PRM en el campo de las proteínas.

5. Experimentos y resultados

La herramienta propuesta fue desarrollada utilizando el lenguaje de programación C++ con QT, la librería de gráficos OpenGL, y el servicio Web DSSP

[8]. La herramienta se ejecutó en una computadora con un procesador Intel Core I5 con 16 gigas de RAM.

La herramienta completa se puede observar en las Figuras 6(a) y 6(b). A continuación se describe cada componente que se encuentra en la herramienta desarrollada:

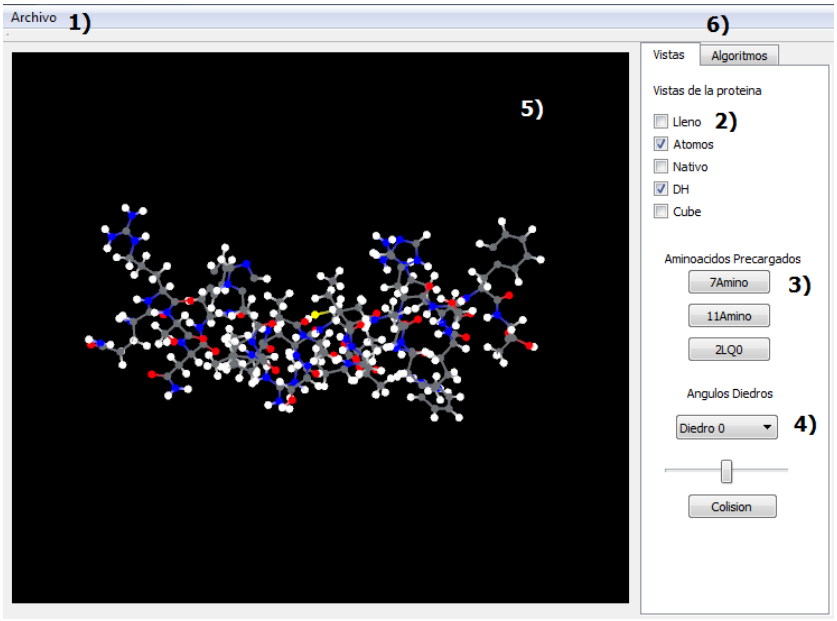
1. En esta sección se pueden abrir proteínas que hayan sido descargadas de la *Protein Data Bank*, permitiendo al usuario utilizar la proteína *monomérica* que desee.
2. Estas casillas de verificación le permiten al usuario modificar la vista actual de la proteína.
3. Estos botones tienen precargadas las proteínas que se utilizaron en este trabajo.
4. En esta sección se puede configurar el estado actual de la proteína modificando los grados de libertad de los distintos ángulos diedros. Con el botón de colisión se puede verificar si la configuración realizada es una configuración correcta.
5. Este lienzo muestra la proteína que está cargada en el sistema dependiendo las casillas que estén verificadas en la sección 2.
6. Estas pestañas nos permite cambiar entre la sección de vistas y algoritmos.
7. Permiten visualizar las configuraciones natural (nativa) y estirada de la proteína, y en el caso del botón invalido encontrar configuraciones no válidas.
8. Son los parámetros básicos necesarios para resolver el problema del plegado de proteínas utilizando PRM.
9. Son los botones que nos permiten inicializar y calcular el *roadmap* con PRM para resolver el problema del plegado de proteínas.
10. Una vez resuelto el *roadmap* se puede visualizar el camino encontrado haciendo clic en el botón animar y modificar la velocidad de la misma modificando la barra horizontal.

Se diseñaron 2 moléculas con 7 aminoácidos ($7Amino$), otra con 11 aminoácidos ($11Amino$) similarmente como se creo la molécula $10 - ALA$ propuesta en [18]. Además se utilizó una molécula más grande de nombre $2LQ0$ que cuenta con 25 aminoácidos.

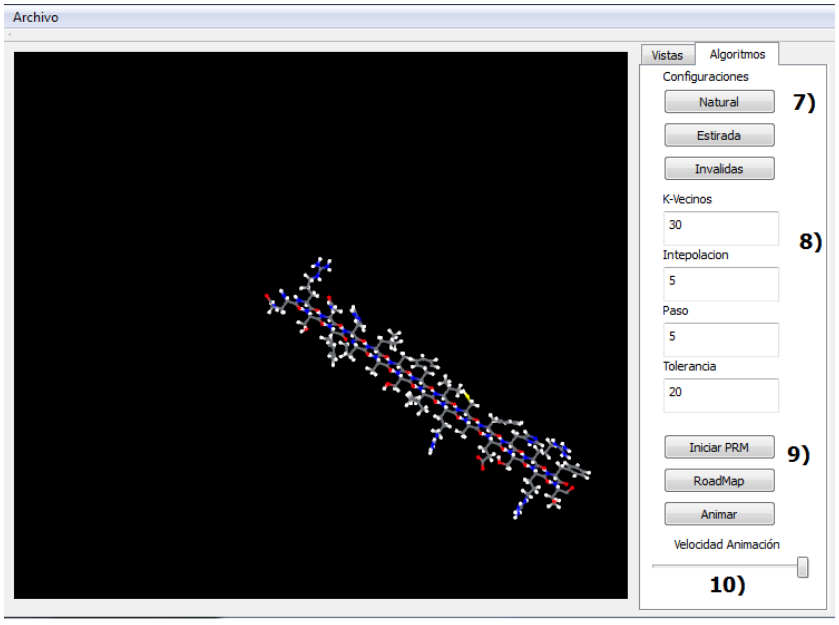
Los parámetros utilizados en el algoritmo de PRM en el problema de plegado de proteína son:

- 30 k-vecinos
- Una interpolación entre configuraciones de 5
- Una distancia euclidiana de 20 para el *Roadmap*[1]
- Un tamaño de paso de 5 para la conexión

Así mismo, se realizaron 5 pruebas con cada una de estas moléculas obteniendo los resultados que se muestran en la Tabla 1.



(a) Esta sección muestra las configuraciones de la vista, y nos muestra la proteína $2LQ0$ en su estado nativo.



(b) Esta sección muestra las configuraciones de algoritmia, se muestra la misma proteína $2LQ0$ en su estado estirado.

Tabla 1. Resultados obtenidos con la herramienta.

Molécula	Nodos	Aristas	Caminos	Tiempo(Horas)
7Amino	722	3107	6	1.5
11Amino	836	47886	4	3.2
2LQ0	2343	107702	3	4.1

6. Conclusiones y trabajo futuro

En este trabajo, se mostró una herramienta para el estudio del plegado de proteínas utilizando técnicas de robótica.

Los resultados al ejecutar dicha herramienta fueron satisfactorios ya que se encontraron diferentes rutas para los procesos de plegado de las diferentes proteínas que se estudiaron. En comparación a las técnicas de Monte Carlo[6,11] que solo obtiene un camino de plegado, las técnicas de PRM [2,18] obtuvieron diferentes caminos de plegado.

Se planea utilizar en trabajos futuros las tecnologías de GPUs para la realización de cálculos y disminuir el tiempo de procesamiento. PRM se adaptó de manera correcta al problema del plegado de proteínas, sin embargo sería importante compararlo con otras aproximaciones basadas en árboles como los RRT.

Actualmente la herramienta permite configurar el diseño molecular de una proteína dada, sería importante en trabajos futuros extender esta visión para trabajar en el ligado de proteínas lo cual es útil en la creación de nuevos fármacos y el diseño de nuevos nanomateriales.

Referencias

1. Amato, N.M., Dill, K.A., Song, G.: Using motion planning to map protein folding landscapes and analyze folding kinetics of known native structures. Journal of Computational Biology 10(3/4), 239–255 (2003)
2. Amato, N.M., Song, G.: Using motion planning to study protein folding pathways. Journal of Computational Biology 9(2), 149–168 (2002)
3. Anfinsen, C.B.: Principles that govern the folding of protein chains. Science 181, 223–230 (1973)
4. de Angulo, V.R., Cortés, J., Siméon, T.: Biocd : An efficient algorithm for self-collision and distance computation between highly articulated molecular models. In: Robotics: Science and Systems. pp. 241–248 (2005)
5. Brändén, C., Tooze, J.: Introduction to Protein Structure. Introduction to Protein Structure Series, Garland Pub. (1999)
6. Covell, D.G.: Folding protein a-carbon chains into compact forms by monte carlo methods. Proteins: Structure, Function, and Bioinformatics 14(3), 409–420 (1992)
7. Doe, R.: Pdb file format @ONLINE (1996)
8. Kabsch, W., Sander, C.: Dictionary of protein secondary structure: pattern recognition of hydrogen-bonded and geometrical features. Biopolymers 22, 2577–637 (1983 Dec 1983)

9. Kavraki, L., Svestka, P., claude Latombe, J., Overmars, M.: Probabilistic roadmaps for path planning in high-dimensional configuration spaces. In: IEEE INTERNATIONAL CONFERENCE ON ROBOTICS AND AUTOMATION. pp. 566–580 (1996)
10. Kavraki, L.E.: Geometric Methods in Structural Computational Biology. (2007)
11. Kolinski, A., Skolnick, J.: Monte carlo simulations of protein folding. ii. application to protein a, rop, and crambin. Proteins 18, 353–66 (1994 Apr 1994)
12. Lansbury, P.: Evolution of amyloid: What normal protein folding may tell us about fibrillogenesis and disease. Proc. Natl. Acad. Sci. USA 96
13. Levitt, M., Gerstein, M., Huang, E., Subbiah, S., Tsai, J.: Protein folding:the endgame. Annual Review of Biochemistry 66(1), 549–579 (1997), pMID: 9242917
14. Munñoz, V., Eaton, W.A.: A simple model for calculating the kinetics of protein folding from three-dimensional structures. Proc Natl Acad Sci U S A 96(20), 113–116 (1999)
15. O'Rourke, J.: Folding and unfolding in computational geometry. In: Akiyama, J., Kano, M., Urabe, M. (eds.) Discrete and Computational Geometry, Lecture Notes in Computer Science, vol. 1763, pp. 258–266. Springer Berlin Heidelberg (2000)
16. Reeke, G.N.: Protein folding: Computational approaches to an exponential-time problem. Annual Review of Computer Science 3(1), 59–84 (1988)
17. Schlick, T.: Molecular Modeling and Simulation: An Interdisciplinary Guide. Springer-Verlag New York, Inc., Secaucus, NJ, USA (2002)
18. Song, G., Amato, N.M.: A motion-planning approach to folding: from paper craft to protein folding. IEEE T. Robotics and Automation 20(1), 60–71 (2004)
19. Thrun, S., Sukhatme, G.S., Schaal, S. (eds.): Robotics: Science and Systems I, June 8-11, 2005, Massachusetts Institute of Technology, Cambridge, Massachusetts. The MIT Press (2005)
20. Zhang, M., Kavraki, L.E.: Solving molecular inverse kinematics problems for protein folding and drug design. In: Currents in Computational Molecular Biology. pp. 214–215. ACM Press, ACM Press (April 2002), book includes short papers from The Sixth ACM International Conference on Research in Computational Biology (RECOM 2002), Washington, DC, 2002

Reviewing Committee

Noé Alejandro Castro-Sánchez
Jair Cervantes
William De La Cruz De Los Santos
Anilu Franco-Arcega
Sofia N. Galicia-Haro
Alexander Gelbukh
David Gonzalez
Miguel Gonzalez-Mendoza
Oscar Herrera
Hector Jimenez Salazar
Asdrúbal López
Lourdes Martínez
Sabino Miranda
Raul Monroy
Antonio Neme
Obdulia Pichardo-Lagunas
Rafael Rojas-Hernández
Grigori Sidorov
Israel Tabarez
Valentín Trujillo Mora
Edgar Vallejo
Nestor Velasco Bermeo

Additional reviewers

Jorge Rodriguez Ruiz
Roberto Alonso
Soujanya Poria
José Camiña
Victor Ferman
Gabriela Ramírez-De-La-Rosa

Impreso en los Talleres Gráficos
de la Dirección de Publicaciones
del Instituto Politécnico Nacional
Tresguerras 27, Centro Histórico, México, D.F.
mayo de 2014
Printing 500 / Edición 500 ejemplares