Un Nuevo Algoritmo de Escala Exterior para el Cálculo de los Testores Típicos

Alexsey Lias-Rodríguez[1], Aurora Pons-Porrata[2]
[1] Departamento de Computación
lias@csd.uo.edu.cu
[2] Centro de Estudios de Reconocimiento de Patrones y Minería de Datos
Universidad de Oriente, Santiago de Cuba, Cuba
aurora@cerpamid.co.cu

Resumen. Los testores típicos tienen gran aplicación en el Reconocimiento de Patrones, especialmente en el problema de Selección de Rasgos. El cálculo de todos los testores típicos a partir de una matriz de entrenamiento tiene una complejidad exponencial con respecto al número de rasgos. Hasta el momento han sido desarrollados varios métodos que agilizan el cálculo de todos los testores típicos, pero actualmente, existen problemas donde este conjunto es imposible de hallar. Con este objetivo, se propone a *BR* como un nuevo algoritmo de escala exterior. Los resultados experimentales demuestran que este método supera notablemente los dos mejores algoritmos reportados en la literatura.

Palabas claves: testores típicos, selección de rasgos

1 Introducción

Uno de los problemas del Reconocimiento de Patrones es la Selección de Rasgos, el cual consiste en encontrar los rasgos que proveen información relevante en el proceso de clasificación. En el Reconocimiento Lógico Combinatorio de Patrones [1] el problema de la Selección de Rasgos se resuelve comúnmente usando la Teoría de Testores [2]. En esta teoría, un *testor* es definido como un conjunto de rasgos que discrimina objetos de diferentes clases. Un testor se llama *irreducible* (*típico*) si ninguno de sus subconjuntos propios es un testor. Cuando nos referimos a testores típicos, nos restringimos a los testores típicos de Zhuravlev's, donde las clases son conjuntos disjuntos, el criterio de comparación entre rasgos es booleano y el criterio de semejanza entre objetos asume que dos objetos son diferentes si al menos uno de sus rasgos también lo son.

Los testores típicos han sido ampliamente utilizados para evaluar la importancia informacional de los rasgos [3] y como conjunto de apoyo en los algoritmos de clasificación [4]. En Minería de Textos se han usado además para la categorización de textos [5] y construcción de resúmenes de documentos [6]. Varios algoritmos han sido desarrollos para calcular el conjunto de todos los testores típicos. Ellos pueden clasificarse de acuerdo con su estrategia computacional en dos categorías: algoritmos de escala exterior y de escala interior. Los primeros calculan los testores típicos

generando los elementos del conjunto potencia en un orden predefinido, pero tratando de evitar el análisis de subconjuntos irrelevantes. Los segundos exploran la estructura interna de la matriz de aprendizaje encontrando condiciones que garantizan la propiedad de testor. En este artículo nos centraremos en la primera estrategia.

La complejidad computacional del cálculo de todos los testores típicos es exponencial respecto al número de rasgos. Actualmente se han desarrollado métodos que aceleran el cálculo de los testores típicos, pero aún existen problemas donde el conjunto de testores típicos es imposible de hallar. Por ello, es de gran importancia desarrollar mejores algoritmos que calculen todos los testores típicos. Los algoritmos de escala exterior *LEX* [7] y *CT-EXT* [8] son los de mejor desempeño reportados en la literatura.

Con este propósito proponemos a *BR*, como un nuevo algoritmo de escala exterior que evita el análisis de un mayor número de subconjuntos irrelevantes y que eficientemente verifica la propiedad de testor tomando como ventaja las operaciones a nivel de bits de la computadora. El nombre del método se debe a las operaciones Binarias y a la estrategia Recursiva que éste utiliza. En este trabajo se extiende el algoritmo *BR* presentado en [9], mediante la introducción de nuevos conceptos (ej. hueco) que aceleran aún más la búsqueda de los testores típicos. Las experimentaciones demuestran que este método supera notablemente los dos mejores algoritmos reportados en la literatura [8].

2 Conceptos Básicos

Antes de presentar nuestro algoritmo, veremos las principales definiciones de la Teoría de Testores y definiremos los conceptos básicos de este método.

Sea M una matriz de aprendizaje que contiene m objetos descritos en términos de n rasgos $\Re=\{X_1,\ldots,X_n\}$ distribuidos en r clases $\{C_1,\ldots,C_r\}$. Cada rasgo X_i toma valores en un conjunto D_i, $i=1,\ldots,n$. Un criterio de comparación de diferencia $\psi_i : D_i \times D_i \rightarrow \{0,1\}$ se asocia a cada X_i (0=similar, 1=disimilar). Aplicando este criterio de comparación para todo posible par de objetos que pertenecen a diferentes clases de M se crea una Matriz Booleana de Diferencia (*MD*). Note que el número de filas en *MD* es $m' = \sum_{i=1}^{r-1} \sum_{j=i+1}^{r} |C_i \| C_j|$, donde $|C_i|$ denota el número de objetos en la clase C_i.

Sean p y q dos filas de *MD*. p es *subfila de* q si en todas las columnas donde p tiene un 1, q también lo tiene. Una fila p de *MD* se llama *básica* si ninguna fila en *MD* es subfila de p. La submatriz de *MD* que contiene todas las filas básicas (sin repeticiones) se llama *matriz básica*. Entonces, un *testor* es un subconjunto de rasgos $\tau=\{X_{i_1},\ldots,X_{i_s}\}$ para los cuales no existe una fila completa de ceros en la matriz resultante luego de eliminar en *MB* todas las columnas correspondientes a los rasgos $\Re\backslash\tau$. τ es un testor típico si no existe un subconjunto propio de τ que tenga la propiedad de testor [2]. Comúnmente, los algoritmos para calcular los testores típicos usan *MB* en vez de *MD* debido a la reducción sustancial de filas.

Sea $(a_1,\ldots,a_u)$ una u-tupla binaria de elementos, $a_i \in \{0,1\}$, $i=1,\ldots,u$. Llamaremos *cardinal de una u-tupla binaria* al número de elementos que ésta contiene (u). La

columna correspondiente a un rasgo X en MB es una u-tupla binaria, cuyo cardinal es el número de filas de MB. Denotaremos esta u-tupla por c_X. También definimos operaciones lógicas sobre u-tuplas binarias de la siguiente manera:

$(a_1, a_2, \ldots, a_u) \vee (e_1, e_2, \ldots, e_u) = (a_1 \vee e_1, a_2 \vee e_2, \ldots, a_u \vee e_u)$

$(a_1, a_2, \ldots, a_u) \wedge (e_1, e_2, \ldots, e_u) = (a_1 \wedge e_1, a_2 \wedge e_2, \ldots, a_u \wedge e_u)$

$\neg(a_1, a_2, \ldots, a_u) = (\neg a_1, \neg a_2, \ldots, \neg a_u)$

(1,…,1) y (0,…,0) representan u-tuplas binarias en las cuales todos sus elementos son unos y ceros, respectivamente.

La notación $[X_1,\ldots,X_s]$, $X_i \in \Re$, representa una lista ordenada de rasgos y *último*$([X_1,\ldots,X_s])$ denota el último elemento de la lista, esto es, X_s. Una lista que no contiene rasgos se denota como []. Llamaremos *longitud de una lista* l, denotada como $|l|$, al número de rasgos que ésta contiene. Todas las operaciones básicas de la teoría de conjuntos (diferencia, intersección, subconjunto o sublista, etc.) pueden ser definidas de forma similar sobre listas ordenadas de rasgos. Con el símbolo + denotaremos la concatenación entre listas ordenadas de rasgos.

Sea l una lista ordenada de rasgos. La notación $[l]$ representa una lista unitaria compuesta por la lista l. En lo adelante, cuando hablemos de lista nos estaremos refiriendo a una lista ordenada de rasgos.

Definición 1. Sea $l = [X_1,\ldots,X_s]$ una lista de rasgos. Llamaremos *máscara de aceptación de* l, denotada como ma_l, a la u-tupla binaria en la cual el i-ésimo elemento es 1 si la i-ésima fila de MB tiene al menos un 1 en las columnas correspondientes a los rasgos de l y es 0, en otro caso.

Definición 2. Sea $l = [X_1,\ldots,X_s]$ una lista de rasgos. Llamaremos *máscara de compatibilidad de* l, denotada como mc_l, a la u-tupla binaria en la cual el i-ésimo elemento es 1 si la i-ésima fila de MB tiene un solo 1 en las columnas correspondientes a los rasgos de l y es 0, en otro caso.

Note que el cardinal de ma_l y mc_l es el número de filas en MB.

Ejemplo 1. Sean $l_1 = [X_1, X_2]$, $l_2 = [X_5, X_6, X_7, X_8, X_9]$ y $l_3 = [X_1, X_2, X_8]$ listas de rasgos de una matriz básica MB. Sus máscaras de aceptación y compatibilidad correspondientes son las siguientes:

$$MB = \begin{pmatrix} X_1 & X_2 & X_3 & X_4 & X_5 & X_6 & X_7 & X_8 & X_9 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \end{pmatrix} \qquad \begin{aligned} ma_{l_1} &= (1,1,0,0,1) \\ ma_{l_2} &= (1,1,1,1,1) \\ ma_{l_3} &= (1,1,0,1,1) \\ mc_{l_1} &= (1,1,0,0,1) \\ mc_{l_2} &= (1,1,0,0,0) \\ mc_{l_3} &= (0,1,0,1,1) \end{aligned}$$

Proposición 1. Una lista de rasgos $l = [X_1,\ldots,X_s]$ es un testor si y sólo si $ma_l = (1,\ldots,1)$.

Definición 3. Sea $l = [X_1,\ldots,X_s]$ una lista de rasgos y $X \in \Re$. Una fila p en MB es un *fila típica de X con respecto l* si ésta tiene un 1 en la columna correspondiente a X y cero en todas las columnas correspondientes a los rasgos en $l \setminus [X]$.

Note que, por definición de testor típico, una lista de rasgos l es un testor típico si l es un testor y satisface la propiedad de tipicidad, esto es, cada rasgo $X \in l$ tiene al menos una fila típica con respecto a $l \setminus [X]$.

Proposición 2. Sean $l = [X_1,\ldots,X_s]$ una lista de rasgos y $X \notin l$ un rasgo de *MB*. La máscara de aceptación de la lista $l + [X]$ se calcula de la siguiente forma:

$$ma_{l+[X]} = ma_l \vee c_X.$$

Proposición 3. Sean $l = [X_1,\ldots,X_s]$ una lista de rasgos y $X \notin l$ un rasgo de *MB*. La máscara de compatibilidad de la lista $l + [X]$ se calcula de la siguiente forma:

$$mc_{l+[X]} = (mc_l \wedge \neg c_X) \vee (\neg ma_l \wedge c_X)$$

Note que las proposiciones 2 y 3 permiten actualizar la máscara de aceptación y compatibilidad, respectivamente cuando un nuevo rasgo se agrega a una lista.

Proposición 4. Sean $l = [X_1,\ldots,X_s]$ una lista de rasgos y $X \notin l$ un rasgo de *MB*. Si al menos una de las siguientes condiciones se cumple:

1. $ma_{l+[X]} = ma_l$
2. $\exists\, X_i \in l$ tal que $mc_{l+[X]} \wedge c_{X_i} = (0,\ldots,0)$

Entonces, X no formará un testor típico con l. En este caso, diremos que X es *excluyente con l*.

La condición 1 significa que X no tiene filas típicas con respecto a l y la segunda indica que X_i pierde todas sus filas típicas debido a X.

Ejemplo 2. X_6 es excluyente con l_1 en el Ejemplo 1, debido a que X_2 satisface $mc_{l_1+[X_6]} \wedge c_{X_2} = (1,0,1,0,1) \wedge (0,1,0,0,0) = (0,0,0,0,0)$. Además, X_8 no es excluyente con l_1, porque $ma_{l_1+[X_8]} \neq ma_{l_1}$ ($l_3 = l_1 + [X_8]$), $mc_{l_3} \wedge c_{X_1} = (0,0,0,0,1)$ y $mc_{l_3} \wedge c_{X_2} = (0,1,0,0,0)$.

Proposición 5. Sean $l = [X_1,\ldots,X_s]$ una lista de rasgos y $X \notin l$ un rasgo de *MB*. $l + [X]$ es un testor típico si y sólo si X no es excluyente con l y $am_{l+[X]} = (1,\ldots,1)$.

La primera condición significa que todos los rasgos de $l+[X]$ tienen al menos una fila típica con respecto a $l+[X]$. La segunda condición garantiza que $l+[X]$ es un testor, por la proposición 1.

Definición 4. Sean $l = [X_1,\ldots,X_s]$ una lista de rasgos, p un entero tal que $1 \leq p \leq s+1$ y $X \notin l$ un rasgo de *MB*. Llamaremos *sustitución de X en l según p*, denotado como *sust*(l, X, p), a la lista $l' = [X_1,\ldots,X_{p-1},X]$. Si $l = []$ entonces *sust*(l, X, 1) = $[X]$.

Note que si $p = s+1$, *sust*(l, X, p) es la lista $l+[X]$.

Definición 5. Sean $l=[X_{i_1},\ldots,X_{i_p}]$ y $l'=[X_{j_1},\ldots,X_{j_q}]$ dos listas de rasgos tal que $l \cap l' = []$. Llamaremos *lista no excluyente de l con respecto a l'*, denotada como *noExcl*(l, l'), a la lista compuesta por los rasgos $X_{i_k} \in l$ tal que X_{i_k} no es excluyente con l' y $l' + [X_{i_k}]$ no es un testor típico.

Ejemplo 3. En la matriz básica del ejemplo 1, *noExcl*(l_2, l_1) = $[X_7,X_8]$. Note que X_5 y X_9 no son excluyentes con l_1, pero $[X_1, X_2, X_5]$ and $[X_1, X_2, X_9]$ son testores típicos.

Definición 6. Sean $l=[X_{i_1},\ldots,X_{i_p}]$ y $l'=[X_{j_1},\ldots,X_{j_q}]$ dos listas de rasgos tal que $l \cap l' = []$. Llamaremos *lista típica de l con respecto a l'*, denotada como *TipL*(l,l') a la lista compuesta por las listas $l' + [X_{i_k}]$ tal que $X_{i_k} \in l$ y $l' + [X_{i_k}]$ son testores típicos.

Ejemplo 4. En la matriz básica del ejemplo 1, $TipL(l_2,l_1) = [[X_1, X_2, X_5], [X_1, X_2, X_9]]$.

Definición 7. Sea $ll = [l_1, l_2,\ldots, l_t]$ una lista de listas de rasgos. Llamaremos *hueco de ll*, y lo denotaremos por h, $1 \leq h \leq t$, al menor índice que cumple que $| l_i | = | l_{i+1} | + 1$, $\forall i = h, h+1,\ldots, t-1$. Llamaremos, además, *eliminación del hueco de ll*, y la denotaremos como $ElimH(ll)$, a la lista $[l_1,\ldots,l_h]$.

3 Algoritmo *BR*

El método propuesto primeramente reordena las filas y columnas de *MB* con el objetivo de reducir el espacio de búsqueda de los testores típicos. La fila con el menor número de 1s y el máximo número de 1s en las columnas de *MB* donde ésta tiene un 1 se coloca como primera fila (ver Pasos 1a y 1b). En el ejemplo 1, la primera y segunda fila tienen dos 1s. La primera se queda en su lugar, porque tiene cuatro 1s en las columnas donde ella tiene un 1 (X_1, X_8). El reordenamiento de las columnas (ver Paso 1c) permite al algoritmo finalizar cuando el rasgo a ser analizado tenga un cero en la primera fila de *MB*. Note que todas las posibles combinaciones de los restantes rasgos nunca serán testores. El reordenamiento de las columnas también intenta reducir la posibilidad de que los rasgos a ser analizados no sean excluyentes con una lista de rasgos, y por consiguiente, minimiza la longitud de las listas de rasgos que deben ser analizadas.

La idea general del algoritmo *BR* es primeramente generar listas de rasgos que satisfagan la propiedad de tipicidad y luego, verificar la condición de testor. Al igual que los algoritmos *LEX* y *CT-EXT*, nuestro método explora el conjunto potencia de los rasgos comenzando por el primer rasgo en *MB* y genera una lista de rasgos candidata a ser testor típico. Una vez que ha sido generada una lista de rasgos candidata, se verifican las propiedades de testor y tipicidad usando las proposiciones 1, 4 y 5. Note que estas proposiciones se basan en las máscaras de aceptación y compatibilidad.

Dada una lista de rasgos candidata *L*, el algoritmo *BR* crea la lista *LP* compuesta por los rasgos X_i que no son excluyentes y no forman un testor típico con *L*. Esto significa que $L+[X_i]$ necesita más rasgos para formar un testor típico. A diferencia de los algoritmos anteriores, los cuales intentan encontrar estos rasgos en *MB*, nuestro método restringe su búsqueda a los rasgos de *LP*. Esto se basa en la siguiente proposición:

Proposición 6. Sean l una lista de rasgos y $X \notin l$ un rasgo de *MB*. Si X es excluyente con l, entonces será también excluyente con toda lista l', tal que $l \subseteq l'$ and $X \notin l'$.

Note que los rasgos X_i que forman un testor típico con *L* no se incluyen en *LP*. En este caso, estos testores típicos se almacenan en *TTR*. Luego, el primer rasgo en *LP* se agrega a *L* y los restantes rasgos de *LP* que no son excluyentes con *L* se seleccionan nuevamente. Este proceso se repite hasta que se encuentren todos los testores típicos que contienen el primer rasgo en *MB*. Luego, el algoritmo comienza a partir del segundo rasgo en *MB* y repite todos los pasos hasta que el primer rasgo a ser analizado tenga un cero en la primera fila en *MB* (ver Paso 3c). Note que el proceso de generación de listas de rasgos candidatas y eliminación de rasgos es recursivo (*TL*

actúa como una pila, en la cual se agregan o se extraen las listas de rasgos con el objetivo de ser reutilizadas en el análisis de nuevas combinaciones de rasgos).

El método propuesto se describe a continuación:

Entrada: Una matriz básica *MB*.
Salida: El conjunto de todos los testores típicos de *MB*.

1. Ordenando filas y columnas en *MB*:
   a. Sea *F* el conjunto de filas que tienen el menor número de 1s.
   b. Para cada fila $f \in F$ obtener el número de 1s en todas las columnas de *MB* que contengan un 1 en *f*. Poner la fila con mayor número como primera fila en *MB*. Si existe más de una fila con el mayor número, tomar cualquiera de ellas.
   c. Sea C^1 (C^0) el conjunto de columnas con un 1 (0) en la primera fila en *MB*. Reordenar las columnas tal que las columnas de C^1 estén a la izquierda y las de C^0 estén a la derecha. Ordenar de forma descendente las columnas de C^1 de acuerdo con el número de 1s. Las columnas en C^0 se ordenan de la misma forma.
2. Inicialización:
   a. $L = []$
   b. Sea *TTR* la lista de testores típicos, $TTR = []$. Note que *TTR* en una lista de listas.
   c. Sea *R* la lista de todos los rasgos en *MB* y $TL = [R]$. Note que *TL* es también una lista de listas.
3. Proceso:
   a. Sea *LR* la última lista de rasgos en *TL*, esto es, $LR = último(TL)$.
   b. Sea *X* el primer rasgo de *LR*.
   c. Si $|TL| = 1$, entonces
      Si la columna correspondiente a *X* (c_X) tiene un cero en la primera fila de *MB*, entonces retornar *TTR* y *finalizar.*
      Si no, si $c_X = (1,\ldots,1)$, entonces $TTR = TTR + [[X]]$, $LR = LR \setminus [X]$ e ir al Paso 3b.
   d. $L = sust(L, X, |TL|)$
   e. Eliminar el último elemento (lista) de *TL*, esto es, $TL = TL \setminus [último(TL)]$.
   f. $LR = LR \setminus [X]$
   g. $LP = noExcl(LR, L)$
   h. $TTR = TTR + TipL(LR, L)$
   i. Si $|LR| > 1$ entonces
      $TL = TL + [LR]$
      Si $|LP| > 1$ entonces $TL = TL + [LP]$
      Si no,
      Sea X_i *el único rasgo de LR.*
      Si $X_i \in último(TTR)$ ó $X_i \in LP$ entonces
      Si existe hueco de *TL* entonces $TL = ElimH(TL)$
   j. Ir al Paso 3.

Note que la lista *LP* incluye los rasgos de *LR* que no son excluyentes con *L* y que no forman un testor típico con *L*, mientras que $TipL(LR, L)$ contiene los rasgos de *LR* que constituyen testores típicos con los rasgos en *L*. Entonces, los Pasos 3g y 3h se realizan simultáneamente de la siguiente manera:

Para cada rasgo *X'* de *LR*:
1. Calcular $ma_{L+[X']}$ a partir de ma_L usando la Proposición 2.
2. Si $ma_{L+[X']} \neq ma_L$ (ver la condición 1 de la Proposición 4) entonces
   a. Calcular $mc_{L+[X']}$ a partir de mc_L usando la Proposición 3.
   b. Si $mc_{L+[X']} \wedge c_{X_i} \neq (0,\ldots,0)\ \forall X_i \in L$ (ver la condición 2 de la Proposición 4) entonces
      Si $ma_{L+[X']} = (1,\ldots,1)$ entonces
      Agregar $L + [X']$ a *TTR* ($L + [X']$ en un testor típico por la Proposición 5)
      si no, Agregar *X'* a *LP*

La condición del Paso 3i ($|LR| > 1$) verifica si es posible seguir agregando listas a *TL* para buscar nuevas combinaciones de rasgos en próximas iteraciones del algoritmo. Note que las listas que se agregan a *TL* siempre tienen al menos dos rasgos. En caso de que *LR* contenga un solo rasgo X_i se verifica si este rasgo forma parte del último testor típico encontrado o si no fue excluyente con *L*. Si se cumplen una de las dos condiciones anteriores, se procede a eliminar de *TL* su hueco si éste existe. Esto evita que el algoritmo explore combinaciones de rasgos (subconjuntos de $L+[X_i]$) que nunca formarán un testor típico. Si X_i forma junto con *L* un testor típico, la eliminación del hueco permite que el algoritmo salte todas esas combinaciones debido a que todo subconjunto de un testor típico no es testor típico. Si se cumple la segunda condición, esto es, que X_i no fue excluyente con *L*, entonces $L+[X_i]$ no es testor y por tanto se saltan todas las combinaciones debido a que todo subconjunto de un no testor es también un no testor.

Ejemplo 5. Sean X_1, X_2, X_3, X_4, X_5 los rasgos de una matriz básica *MB*. En la tabla de la izquierda se representan las iteraciones del algoritmo y los diferentes valores que van tomando las listas *L*, *TL*, *LR* y *LP*. La tabla de la derecha muestra el orden de generación de las listas de rasgos sin considerar el concepto de hueco (eliminando la parte "Si no" del paso 3i) y considerándolo. En ellas, una lista de rasgos $[X_1,X_2\ldots,X_s]$ se representa de forma compacta como [12…*s*]. Las listas de rasgos que constituyen testores típicos se resaltan en negritas.

Iteración	***L***	***TL***
1	[]	[[1 2 3 45]]
2	[1] ⟵ [[2 345], (*LR*) [2345]] (*LP*)	
3	[1 [[2 345], 2] ⟵ [345], (*LR*) [345]] (*LP*)	
4	[1 [[2 345], 2 [345], 3] ⟵ [45], (*LR*) [45]] (*LP*)	
5	[1 [[2345], 2 [345], } Se eliminan 3 [45], } 4] ⟵ [5]] } (*LR*)	
6	[[2345]]	

$$MB = \begin{pmatrix} X_1 & X_2 & X_3 & X_4 & X_5 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \end{pmatrix}$$

Sin hueco		**Con hueco**	
[1]	[1235]	[1]	[1234]
[12]	**[12345]**	[12]	[1235]
[13]	[1245]	[13]	**[12345]**
[14]	[134]	[14]	[2]
[15]	[135]	[15]	
[123]	[1345]	[123]	
[124]	[145]	[124]	
[125]	[2]	[125]	
[1234]			

Note en la tabla de la izquierda que la lista *TL* en la iteración 5 del algoritmo es [[2345], [345], [45], [5]]. En este caso, el hueco de *TL* es 1 (a partir de la primera lista de *TL* se cumple la definición 7) y *ElimH*(*TL*)=[[2345]]. Como puede observarse en la tabla de la derecha, la introducción del concepto de hueco evita el análisis de 5 subconjuntos de rasgos (aparecen sombreados en la tabla). Note que todos ellos son subconjuntos de un testor típico encontrado ([12345]).

Las características que permiten al algoritmo *BR* evitar el análisis de subconjuntos irrelevantes de rasgos se muestran a continuación:

- El algoritmo examina directamente las combinaciones de rasgos que se produce entre los rasgos de *LP* que no son excluyentes con *L*, evitando el análisis de las restantes combinaciones.
- Debido a que los rasgos que forman un testor típico con *L* no son incluidos en *LP*, el algoritmo descarta todos los supraconjuntos de los testores típicos hallados.
- Al eliminar el hueco de la lista *TL* se evita el análisis de combinaciones de rasgos que son subconjuntos de un testor típico o subconjuntos de un no testor.

El algoritmo *CT-EXT* primeramente genera los testores y luego verifica la propiedad de tipicidad, mientras que los algoritmos *LEX* y *BR* primeramente generan los conjuntos de rasgos que satisfacen la propiedad de tipicidad y luego, verifican la condición de testor. *CT-EXT* intenta reducir el costo de verificación de la propiedad de tipicidad que el *LEX* realiza, pero genera un mayor número de subconjuntos de rasgos.

Ejemplo 6. La siguiente tabla muestra los subconjuntos de rasgos generados por los algoritmos *LEX*, *CT-EXT* y *BR* para una matriz básica hasta que el conjunto $\{X_2, X_6\}$ (representado como 26) sea generado. Los testores típicos se resaltan en negritas y del *LEX* y el *CT-EXT* se sombrearon los subconjuntos que el *BR* no generó. Como podemos observar, *BR* genera el menor número de subconjunto de rasgos. Note que $LP=[X_3, X_4]$ cuando los primeros 6 subconjuntos son generados. Por lo tanto, *BR* puede saltar a $\{X_1, X_3, X_4\}$ ignorando las restantes combinaciones que incluyen a X_1. Sin embargo, *LEX* no es capaz de ignorar estas combinaciones. Por otro lado, *CT-EXT* examina varios subconjuntos incluyendo a X_1 y X_2 aún cuando ninguno de ellos constituyen un testor típico. La definición de que un rasgo *X* contribuya a un subconjunto [8] sólo verifica que *X* tenga al menos una fila típica, pero ignora que los rasgos en los subconjuntos pueden perder sus filas típicas debido a *X*. Note que X_2 contribuye a $\{X_1\}$, pero X_1 pierde su fila típica (la primera) debido a X_2.

LEX		*CT-EXT*			***BR***	
1	146	1	135	**23**	1	**23**
12	**15**	12	136	24	12	24
13	**16**	123	14	**25**	13	**25**
134	2	124	145	**26**	14	**26**
135	**23**	125	146		**15**	
136	24	126	**15**		**16**	
14	**25**	13	**16**		**134**	
145	**26**	**134**	**2**		**2**	

$$MB = \begin{pmatrix} X_1 & X_2 & X_3 & X_4 & X_5 & X_6 \\ 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 1 & 1 \end{pmatrix}$$

4 Experimentación

Con el objetivo de evaluar el funcionamiento del método propuesto para calcular todos los testores típicos, comparamos el tiempo de ejecución de nuestro método con los dos mejores algoritmos reportados en la literatura: *LEX* y *CT-EXT*. Es importante mencionar que el código fuente del algoritmo *CT-EXT* fue proporcionado por los autores. Para asegurar una comparación justa todos los métodos fueron ejecutados en un Intel Pentium Dual Core 1.6 GHz, 1 Gb RAM.

Para esta comparación se usaron cinco conjuntos de datos obtenidos del repositorio UCI[1]. Para cada uno se generaron las matrices básicas usando la igualdad estricta como criterio de comparación para todos los rasgos. La tabla 1 muestra el tiempo de ejecución de los métodos para las matrices básicas del conjunto de datos reales y dos matrices básicas generadas aleatoriamente. Note que estas matrices tienen diferentes dimensiones (vea la Columna 3). La última columna (NTT) indica el número de testores típicos calculados.

Tabla 1. Tiempo de ejecución (h:m:s:ms) de los algoritmos para varias matrices básicas.

Data set	Class	*BM*	*LEX*	*CT-EXT*	*BR*	NTT
Zoo (101 x 17)	7	14 x 17	0:0:00:15	0:0:0:718	0:0:00:00	34
Mushroom (8124 x 22)	2	30 x 22	0:0:00:16	0:0:0:750	0:0:00:00	292
Chess (3196 x 36)	2	29 x 36	0:2:22:16	0:8:01:67	0:0:00:12	4
Dermatology (366 x 34)	6	1124 x 34	0:25:45:7	1:43:15:6	0:0:58:22	115556
Promoter (106 x 57)	2	2761 x 57	1:07:27:5	4:24:23:8	0:3:18:51	7456943
Random	-[2]	150 x 70	0:55:45:3	2:06:30:4	0:4:02:67	44165054
Random	-[2]	100 x 100	2:22:01:9	> 20 hrs	0:10:30:1	183051234

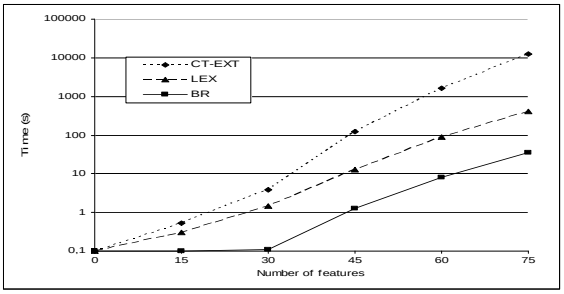

Figura. 1. Tiempo de ejecución (en segundos) para matrices básicas de 50 filas variando el número de rasgos.

Como podemos observar, mientras mayor es la dimensión de la matriz básica, mayor es el tiempo necesario para calcular los testores típicos en todos los algoritmos. Es importante notar también que *BR* logra considerables reducciones de tiempo con respecto a *LEX* y *CT-EXT*. A diferencia de los resultados reportados en [8], nuestros experimentos revelan que el *CT-EXT* funciona peor que el *LEX*.

Con el objetivo de estudiar el comportamiento de los algoritmos, en la Figura 1 se muestra el tiempo de ejecución (en segundos) para matrices básicas de 50 filas variando el número de rasgos de 15 a 75. Como se esperaba, el tiempo de todos los métodos crece exponencialmente cuando el número de rasgos se incrementa. Sin embargo, note que nuestro método se ejecuta aproximadamente 10 veces más rápido que el mejor competidor, el *LEX*, y alrededor de 100 veces más rápido que el *CT-EXT*. Así, podemos concluir que el algoritmo *BR* es significativamente más eficiente que los otros algoritmos.

[1] http://archive.ics.uci.edu/ml/

[2] El número de clases se ignora, porque se generaron aleatoriamente matrices de 0s y 1s.

5 Conclusiones

En este artículo se propuso a *BR* como un nuevo algoritmo de escala exterior para el cálculo de todos los testores típicos a partir de una matriz de aprendizaje. Los resultados experimentales demuestran que este método supera significativamente los dos mejores algoritmos reportados en la literatura. Las principales contribuciones que aseguran la aceleración en el cálculo de todos los testores típicos son: un nuevo método para verificar las propiedades de tipicidad y testor, las cuales se basan en la lógica binaria y aprovecha las operaciones a nivel de bits de la computadora, la introducción de un mecanismo para generar subconjuntos de rasgos candidatos que evita el análisis de un mayor número de subconjuntos irrelevantes, y el orden previo a la matriz básica que garantiza que el método finalice lo antes posible.

Como trabajo futuro proponemos extender nuestro método con el objetivo de obtener otras generalizaciones de los testores típicos no restringidos a los testores de Zhuravlev's (e.j. ε-testors y testores difusos [2]). También proponemos realizar nuevas experimentaciones con matrices básicas de diferentes densidades para evaluar el funcionamiento del método propuesto.

Referencias

1. Martínez-Trinidad, J.F., Guzmán-Arenas, A.: The Logical Combinatorial approach to Pattern Recognition: an overview through selected Works. Pattern Recognition 34(4), 741--751 (2001).
2. Lazo-Cortés, M., Ruiz-Shulcloper, J., Alba-Cabrera, E.: An overview of the evolution of concept testor. Pattern Recognition 34(4), 753--762 (2001).
3. Ortiz-Posadas, M.R., Martínez-Trinidad, J.F., Shulcloper, J.R.: A new approach to diferential diagnosis of diseases. Int. J. Biomed. Compu. 40(3), 179--185 (1996).
4. De la Vega-Doria, L.A., Carrasco-Ochoa, J.A., Shulcloper, J.R.: Fuzzy KORA-Ω algorithm. In: 6th European Congress on Intelligent Techniques and Soft Computer, pp. 1190--1194. Aachen, Germany (1998).
5. Pons-Porrata, A., Gil-García, R., Berlanga-Llavori, R.: Using Typical Testors for Feature Selection in Text Categorization. In: CIARP 2007. LNCS, vol. 4756, pp. 643--652 (2007).
6. Pons-Porrata, A., Ruiz-Shulcloper, J., Berlanga-Llavori, R.: A Method for the Automatic Summarization of Topic-Based Clusters of Documents. In: CIARP 2003. LNCS, vol. 2905, pp. 596--603 (2003).
7. Santiesteban-Alganza, Y., Pons-Porrata, Aurora: LEX: a New Algorithm for Computing Typical Testors (in Spanish). Revista Ciencias Matemáticas 21(1), 85--95 (2003).
8. Sánchez Díaz, G., Lazo Cortés, M.: CT-EXT: An Algorithm for Computing Typical Testor set. In: CIARP 2007. LNCS, vol. 4756, pp. 506--514 (2007).
9. Lias-Rodríguez, Alexsey and Pons-Porrata, Aurora: BR: A New Method for Computing all Typical Testors. In: CIARP 2009. LNCS, vol. 5856, pp. 379-386 (2009).