Espacios de Versiones y Master Mind.
Un ejemplo de los efectos de la representación conceptos y el ruido en los ejemplos en el Aprendizaje de Clasificadores

Salvador De La Torre C.
Estudiante de Maestría en Sistemas
Universidad Nacional de Colombia, Departamento de Sistemas,
Bogotá D.C., Colombia
Sdelator@etb.net.co

Sandra Liliana Rojas Martínez
Profesora Departamente de Sistemas
Universidad Nacional de Colombia, Departamento de Sistemas,
Bogotá D.C., Colombia
Slrojasm@unal.edu.co

Resumen

El *Espacios de Versiones* y su *Algoritmo de Eliminación de Candidatos* son una solución conocida al problema del aprendizaje de clasificadores por inducción de gran importancia teórica que en la práctica se ve restringida por su intolerancia al ruido en el entrenamiento y un crecimiento combinatorio en la cantidad de datos almacenados. *MasterMind* es un juego de información perfecta pero incompleta que no permiten la aplicación directa de dicho algoritmo. En este artículo se muestra la aplicación de las técnicas de manejo de ruido de Norton y Hirsh en un algoritmo basado en Espacios de Versiones capaz de jugar al Master Mind con destreza y se miden los ahorros en la memoria consumida producto de la detección temprana de contradicciones. También se analiza el efecto de una posterior representación de conceptos diseñada a la medida en el consumo de memoria y en la estrategia de juego.
Palabras claves: Aprendizaje de clasificadores, espacios de versión, master mind, representación de conceptos, LISP.

Abstract

The Version Space and its Candidate Elimination algorithm are a well known solution to the problem of inductive classifier learning. Of great theoretical importance, in practice it is limited by its intolerance to noisy training examples and a combinatorial growth in the amount of stored data. Mastermind is a game of perfect but incomplete information that does not allow the direct application of the aforementioned algorithm. This article shows the application of the noise handling techniques of Norton and Hirsh on a version space based algorithm capable of playing Master Mind skillfully and measures the memory savings obtained by the early detection of contradictions. It also discusses the effect of a subsequent specifically tailored representation of concepts on memory consumption and game strategy.
Key words: classifiers learning, version spaces, mastermind, concept representation, LISP.

1. Los Espacios De Versiones

El aprendizaje de conceptos usando Espacios de Versiones y el Algoritmo de Eliminación de Candidatos es un método simbólico que busca escoger entre un conjunto potencialmente grande de clasificadores candidatos a aquel cuyos resultados concuerden con los de una serie de casos conocidos llamados *ejemplos* [5]. Todos los clasificadores son bulianos por lo que solo son capaces de evaluar si un caso cumple o no con una propiedad o característica. Un *espacio de versiones* es un conjunto de conceptos expresados en un lenguaje L que son consistentes con todos los ejemplos de entrenamiento observados hasta un momento dado, es decir, con un subconjunto de los ejemplos disponibles. Los ejemplos de entrenamiento constan de un caso particular al que se le asocia un valor buliano y juntos deben ser transformables en un concepto que se pueda expresar en el leguaje L. Se puede definir así al espacio de versiones EV como una función de los ejemplos analizados J dejando implícito el lenguaje L: $EV(J)$ es el conjunto más grande posible de conceptos expresados en un lenguaje L consistente con todos los ejemplos del conjunto J. Si el conjunto J es inconsistente $EV(J)$ será vació y en este casó se dirá que el espacio de versiones colapsó.

El *Algoritmo de Eliminación de Candidatos* calcula $EV(J)$ para un conjunto o secuencia de ejemplos J partiendo desde un espacio de versiones inicial EV_0 que contiene a todos los conceptos posibles en L. El algoritmo guarda cierta similitud con la búsqueda de raíces en funciones continuas por el método de la bisección. Para el *Algoritmo de*

Eliminación de Candidatos el espacio de versiones se define completamente mediante dos conjuntos extremos de conceptos: el conjunto de los conceptos más generales y el conjunto de conceptos más específicos, ambos consistentes con los ejemplos observados por lo que a su vez son subconjuntos del espacio de versiones al que representan; esto implica que se debe poder establecer un orden al menos parcial entre los conceptos formando un *espacio de conceptos*. Se define el orden "$\preceq$" sobre el conjunto de conceptos de forma que para dos elementos cualesquiera c_1 y c_2 de este conjunto se tenga que $c_1 \preceq c_2$ solo si c_1 es más específico o igual a c_2. Si ***G*** y a ***E*** son los conjuntos de conceptos respectivamente más generales y más específicos consistentes con los ejemplos observados, se debe cumplir siempre que:

$$(c \in EV) \Leftrightarrow (\exists c_1 \in E \subseteq EV)(\exists c_2 \in G \subseteq EV)(c_1 \preceq c \preceq c_2)$$

Con el *Algoritmo de Eliminación de Candidatos* cada ejemplo positivo generaliza a *E* y cada ejemplo negativo especializa a *G*; el resultado es que los dos conjuntos se acercan descartando buena parte de los conceptos candidatos hasta que, de contarse con ejemplos suficientes, ambos *convergen* en un único concepto ó hasta que *EV* se hace vació, o lo que es igual, *colapsa*.

1.2 El Lenguaje de las Conjunciones de Literales Positivos Representado Como N-Tuplas

No todos los lenguajes formales son aptos para ser usados con el *Algoritmo de Eliminación de Candidatos*, en particular los lenguajes que incluyen disyunciones no admiten una directa generalización mínima de los conceptos más específicos que los acerque a los conceptos más generales [1]. Los *lenguajes de conjunciones de literales positivos* sí permiten formular conceptos fáciles de generalizar o especializar mínima y convergentemente y por esta razón son los más empleados en los libros introductorios tocantes al tema [4]. También es una costumbre difundida el emplear solo *conceptos* positivos, es decir aquellos que al evaluarse positivos con un caso lo clasifican como positivo. Es habitual decir que un concepto acepta o cubre un caso cuando lo clasifica positivamente y que lo rechaza o no lo cubre en caso contrario.

Las conjunciones de literales positivos se puede visualizar como una *n-tupla* siendo *n* el número de categorías o características de los objetos a clasificar y cada literal un valor posible que aparecerá en la tupla solo en la posición que corresponde a su categoría [9]. Se pueden definir las *dimensiones* D_i como el conjunto de valores o literales de la ranura *i* sin incluir los valores especiales "*" y "*?*"."*" se interpreta como "todos los valores" y "*?*" como "valor indeterminado". Cada D_i agrupa literales de una misma categoría que ocupan la misma posición en las distintas túplas de ejemplos y conceptos. Un concepto *c* pertenecerá al leguaje de las n-tuplas *L* si $c = (x_1, x_2, \ldots, x_n) \wedge \forall i \in N\ (1 \le i \le n) \Rightarrow ((x_i \in D_i \cup \{*, ?\}) \wedge (*, ? \notin D_i))$ con *n* el número de características o categorías distintas que se requieran. Se asume que *c* es un concepto positivo. En *L* los valores de los conjuntos límites iniciales se definen como G_o = {((*,*,*,*), *verdadero*)} y E_o = {((?,?,?,?), *verdadero*)} que corresponden al espacio de versiones inicial EV_o el cual no ha recibido ningún ejemplo de entrenamiento. Definir a E_o así evita tener que iniciar con un ejemplo positivo sin afectar la capacidad de aprendizaje.

2. Espacios de Versiones a prueba de ruido, incoherencias e información incompleta

Diferentes enfoques han sido propuestos para darle al Algoritmo de Eliminación de Candidatos tolerancia al ruido. El ruido produce inconsistencias que generalmente llevan a una convergencia temprana y a un posterior colapso, ó a un concepto errado. Mitchel propuso llevar una gran lista de todos lo espacios de versiones antes y después de cada actualización para todos los posibles conjuntos de entrenamiento que incluyesen a lo sumo a los ejemplos considerados hasta el momento. Una alternativa más simple y efectiva fue propuesta por Norton y Hish siempre que se pueda modelar el ruido presente en los ejemplos de entrenamiento de la forma que se explicará enseguida [6]. Este método presenta un crecimiento exponencial pero usualmente menor al del método de Mitchel. Adicionalmente las pruebas empíricas sugieren que muchas de las hipótesis invalidas tardan poco en ser eliminadas por colapsos de sus espacios reduciendo notoriamente dicho crecimiento.

2.1 Generación de múltiples Espacios de Versiones

Para cada ejemplo observado *o* se puede asumir que existen uno o más ejemplos reales $e_1, e_2, \ldots, e_k$, $k \in N$ que pudieron ser transformados en *o* tras ser afectados por el ruido cada uno con una probabilidad $P(o|e_i)>0$ para $1 \le i \le k$, $i \in N$ y *k* el número máximo de alternativas o hipótesis a considerar. Para cada hipótesis se necesitará un *espacio de versiones probabilístico* propio que es un espacio de versiones normal con una probabilidad asociada *p* y que será denotado como *EVP*. Al conjunto de todos los espacios de versiones probabilísticos que cubren las distintas hipótesis o ejemplos reales posibles se le llamará *CEVP*.

Extendiendo el efecto del ruido desde los ejemplos individuales, un conjunto de ejemplos reales $J_m = \{e_1, e_2, \ldots\}$ se podría transformar por efectos del ruido en un conjunto de ejemplos observados $X = \{o_1, o_2, \ldots\}$. [6] utilizan secuencias en lugar de conjuntos por que si bien el orden de los ejemplos es indiferente para el *Algoritmo de Eliminación de Candidatos*, es más fácil establecer en las primeras la relación entre cada uno de los ejemplos reales $e_i \in S_m$ y los observados $o_i \in O$ haciendo que la secuencia real $S_m = (e_{m1}, e_{m2}, \ldots)$ se relacione con la secuencia observada $O = (o_1, o_2, \ldots)$ solo si cada ejemplo real e_{mi} y su correspondiente observación o_i cumplen que $P(o_i|e_{mi}) > 0$. Además la naturaleza secuencial de los algoritmos de computadora hace que los ejemplos se procesen uno a la vez

en un orden determinado aunque quizás carente de significado. Téngase en cuenta que es muy posible que la secuencia O cumpla esta correspondencia con varias secuencias reales distintas S_1, S_2,... .

[6] emplean la estrategia *Bayesiana* de *máximo a posteriori* para elegir el mejor concepto c_i entre todos los conceptos posibles del *CEVP*. Según este enfoque el mejor concepto es aquel con mayor probabilidad de ocurrencia dada la secuencia observada O denotada por $P(c_i|O)$; hay que tener en cuenta que no se conoce la secuencia real con certeza y por ello se menciona a O y no a S. Podría presentarse más de un concepto con probabilidad máxima. [6] hacen los siguientes supuestos para facilitar el cálculo de este valor: a) Se conoce un modelo del ruido que proporciona la probabilidad $P(o_i|e_{mi})$. b) Todos los conceptos expresados en el lenguaje L tienen una misma probabilidad apriori: $\forall c_i \in L \ \forall c_j \in L \ (P(c_i) = P(c_j))$. c)El concepto a aprender es determinístico y por lo tanto consistente con los ejemplos de entrenamiento reales, por lo que si el concepto $c \in L$ no es congruente con todos los ejemplos reales de la secuencia S se debe tener que $P(c|S) = 0$ y viceversa.

Defínase la función $EV(S)$ como aquella que retorna el conjunto más grande de conceptos en L consistente con la secuencia de ejemplos S, tal como había hecho ya anteriormente, y que se puede calcular con el *Algoritmo de Eliminación de Candidatos*; entonces se cumple que [6]:

$$P(c_j \mid O) \propto \sum_{\substack{j \\ c_i \in EV(S_j)}} P(O \mid S_j)$$

Es posible definir el conjunto de espacios que cubren las diferentes secuencias reales probable o hipótesis compatibles con lo observado como una función de la secuencia observada:

$$CEVP(O) = \{ EVP(S_j) = (P(O|S_j), EV(S_j)) : EV(S_j) \subseteq EV_0 \wedge P(O|S_j) > 0\}$$

Para calcular el valor proporcional a la probabilidad de un concepto c_i o pseudo-probabilidad solo es necesario considerar las secuencias S_j que posiblemente transmutaron en O y que generan además *espacios de versiones* que contienen a dicho concepto. Se puede aprovechar la naturaleza incremental del *Algoritmo de Eliminación de Candidatos* y realizar el proceso con cada ejemplo observado uno por uno, con lo que se tendrá que considerar los varios ejemplos reales que potencialmente generaron cada observación, dando así a lugar a un número total de hipótesis en $CEVP(O)$ igual al número de ejemplos reales para el ejemplo observado en curso multiplicado con el número de hipótesis acumuladas previamente. El proceso incremental cumple con las siguientes ecuaciones:

$$EV((e_{j1}, e_{j2}, \dots , e_{jk}, e_{j(k+1)})) = EV((e_{j1}, e_{j2}, \dots , e_{jk})) \cap EV((e_{j(k+1)}))$$

$$|CEVP((o_1, o_2, \dots , o_k, o_{k+1}))| = |CEVP((o_1, o_2, \dots , o_k))| \times |CEVP((o_{k+1}))|$$

$$P((o_1, o_2, ..., o_k, o_{k+1})|(e_{j1}, e_{j2}, \dots , e_{j(k+1)})) = P(o_1|e_{j1}) \cdot P(o_2|e_{j2}) \cdot \dots \cdot P(o_k|e_{jk}) \cdot P(o_{k+1}|e_{j(k+1)})$$

$$= P((o_1, o_2, ..., o_k)|(e_{j1}, e_{j2}, \dots , e_{jk})) \cdot P(o_{k+1}|e_{j(k+1)})$$

Nótese que estas formulas no tienen en cuenta el posible colapso del espacio de versiones asociado, lo cual podría ser o no importante según las características del problema.

Si los espacios de versiones en $CEVP(O)$ son todos mutuamente disyuntos, para cualquier concepto c_i en L solo existirá a lo sumo un único espacio de versiones $EV(S_j)$ que lo contenga a y en estas condiciones $P(c_i|O) \propto P(O|S_j)$ y todos los conceptos de $EV(S_j)$ tendrán la misma prioridad a posteriori; para hallar los conceptos más probables bastará con encontrar el $EVP(S_j)$ más probable. Si los espacios no son disyuntos es necesario encontrar las intersecciones mutuas no vacías entre los distintos espacios de versiones del $CEVP(O)$ hasta hallar la intersección con la máxima probabilidad; el espacio de versiones resultante tendrá una probabilidad proporcional a la suma de las probabilidades de los espacios de versiones involucrados. Sea $EVM(O)$ el espacio de versiones que contiene los conceptos con la mayor probabilidad dada la secuencia observada O, con $CEVP(O)=\{EVP(S_1), EVP(S_2), \dots , EVP(S_m)\}$ y sea $m=|CEVP(O)|$; entonces:

$$(d \in EVM(O)) \quad \Leftrightarrow (\exists EVP(S_i) \in CEVP(O) \ ((EVP(S_i)=(p_i, EV(S_i))) \wedge (d \in EV(S_i))) \wedge$$
$$(\forall EVP(S_j) \in CEVP(O)) \ (\forall c \in EV(S_j)) \ ((EVP(S_j)=(EV(S_j), p_j)) \Rightarrow (P(d|O) \geq P(c|O))))$$

$$\exists A \subseteq CEVP(O) \ (A \neq \{\ddot{o}\}) \ \forall i \in N \cap [1,|A|] \left((EV(S_i), p_i) \in A\right) \left(EVM(O) = \bigcap_{i=1}^{|A|} EV(S_i) \right)$$

3. Master Mind y el Juego de las Picas & Fijas

Master Mind es un juego para dos jugadores originalmente inventado en 1970 por Mordecai Meirowitz cuyo objetivo es que alguno de los jugadores adivine el "código" o clave del otro en el menor número de jugadas posible mediante intentos sucesivos para cada uno de los cuales recibe del jugador contrario una información veraz pero limitada. Si se deja a un lado los detalles de los colores y tamaños de las fichas y la organización del tablero se le puede describir como un juego numérico conservando su esencia por completo. Un jugador al que se llamará "*oráculo*" elige aleatoriamente o a su gusto un número de cuatro cifras en base seis que permanecerá invariable a lo largo de todo el juego; si se representa este número como una cuádrupla de las cifras individuales del número elegido se podrá decir que es de la forma *(a, b, c, d)* tal que $a, b, c, d \in \{0, 1, 2, 3, 4, 5\}$; a esta cuádrupla se le denominará *código*. El jugador contrario llamado "*adivinador*" realiza máximo 12 intentos en cada uno de los cuales le comunicará al oráculo códigos de cuatro cifras en base 6; a cada intento el oráculo responderá con un *resultado* compuesto por dos valores: el número de cifras que ambos códigos, el del oráculo y el del adivinador, tienen en

común en las mismas posiciones dentro de la cuádrupla, y el número de cifras iguales que están en posiciones distintas en ambas cuádruplas; a los primeros se les llamará *fijas* y a los últimos *picas*.
El juego de Pics&Fijas es una variación del juego de Master Mind en el que cada cifra puede ir del 0 al 9, es decir que el código está en base 10, pero no pueden repetirse cifras ni en el número a adivinar ni en los intentos que se presentan al oráculo. Por ejemplo el número 1231 no es un código válido a adivinar ni un intento plausible. El siguiente es un posible juego completo de ejemplo:
Código elegido por el oráculo: (5, 2, 9, 0)
Intento #1 del adivinador: (1, 2, 3, 4); el oráculo retorna: fijas = 1, picas = 0.
Intento #2 del adivinador: (1, 7, 9, 8); el oráculo retorna: fijas = 1, picas = 0.
Intento #3 del adivinador: (1, 0, 5, 6); el oráculo retorna: fijas = 0, picas = 2.
Intento #4 del adivinador: (0, 2, 9, 5); el oráculo retorna: fijas = 2, picas = 2.
Intento #5 del adivinador: (5, 2, 9, 0); el oráculo retorna: fijas = 4, picas = 0.
Fin del juego en 5 jugadas

3.1 Picas & Fijas y los Espacios de Versiones

Una forma muy directa de representar el juego como conceptos de espacios de versiones viene del uso de cuádruplas tal como se planteo anteriormente. Se puede definir un lenguaje L con las dimensiones $D_1=D_2=D_3=D_4=\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$ y los símbolos especiales para los cuales $? \prec 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 \prec *$; es la relación "más específico que". Con este lenguaje se podría especificar por ejemplo que el primer valor del código es con certeza el número 5, pero que se desconocen los demás valores, mediante el ejemplo de entrenamiento ((5, *?* , *?* , *?*), *verdadero*); ó se podría decir que toda cuádrupla que finalice con un cuatro es falsa, es decir que el código no tiene un 4 en su última posición, a través del ejemplo ((*, *, *, 4), *falso*). También se podría indicar que todos las cuádruplas son verdaderas con ((*, *, *, *), *verdadero*), aunque esto último nunca pasaría en un juego de Picas&Fijas real.

3.1.1 Ejemplos de Entrenamiento

Generar todos los ejemplos que se deducen de los resultados de un intento se limita a calcular las permutaciones y combinaciones de las posiciones y valores de la cuádrupla de dicho intento. Se puede empezar por especular cuales son las *fijas* de la cuádrupla, si las hay, las cuales serán las combinaciones de todos los valores de la tupla con un tamaño igual al número de fijas. Para cada hipótesis existirá un concepto que afirme en cada caso que los valores escogidos como fijas están ya en el lugar correcto dentro de la cuadrupla y que no asevere nada acerca de las demás posiciones. La siguientes funciones se definen para n-tuplas para que sirvan en variantes del Picas & Fijas de varias cifras aunque en la versión tradicional de este juego $n=4$.

- Combinaciones-F(*n-tupla*, *número-de-fijas*) → *Conjunto-de-n-tuplas*

$D \in$ *Combinaciones-F*$(C, k) \Leftrightarrow (D=(d_1, d_2, ..., d_n)) \wedge (C=(c_1, c_2, ..., c_n)) \wedge \exists F \subseteq N \cap [1, n]\ (|F|=k)\ \forall i \in N \cap [1, n]$
$((i \in F \Rightarrow d_i=c_i) \wedge (i \notin F \Rightarrow d_i=?))$

Se cumple que $|Combinaciones\text{-}F(C, f)| = \binom{|C|}{f} = \frac{|C|!}{(|C|-f)! \cdot f!}$.

Ejemplos: *Combinaciones-F*((1, 2, 3, 4), 1) = {(1, *?*, *?*, *?*), (*?*, 2, *?*, *?*), (*?*, *?*, 3, *?*), (*?*, *?*, *?*, 4)}; *Combinaciones-F*((1, 2, 3, 4), 3) = {(1, 2, 3, *?*), (1, 2, *?*, 4), (1, *?*, 3, 4), (*?*, 2, 3, 4)}
Un tratamiento similar se aplica a las posibles *picas* de las cuales se sabría el valor y que deberían estar ubicadas en posiciones distintas a las que ocupaban dentro del intento. Puesto que no solo importa qué valores se asumen como picas si no también sus nuevas posiciones, se necesita calcular las permutaciones de las mismas que no repitan ninguna de las posiciones originales anteriores. En adelante se entiende que $a \in C$ con $C=(c_1, c_2, ..., c_n)$ si y solo si existe al menos un entero i entre 1 y n tal que $c_i=a$.

- Permutaciones-P-F(*n-tupla-intento, n-tupla-fijas*, *número-picas*) → *Conjunto-de-n-tuplas*

$D \in$ *Permutaciones-P-F* $(C, F, l) \Leftrightarrow (D=(d_1, d_2, ..., d_n)) \wedge (C=(c_1, c_2, ..., c_n)) \wedge (F=(f_1, f_2, ..., f_n)) \wedge \exists P \subseteq N \cap [1, n]$
$(|P|=l)\ \forall i \in N \cap [1, n](i \in P \Rightarrow ((f_i=?) \wedge (d_i \in C) \wedge (d_i \notin F)\ \wedge (d_i \neq c_i))) \wedge (i \notin P \Rightarrow (d_i=f_i)))$

La función *Permutaciones-P-F* requiere como su segundo parámetro la cuádrupla de un concepto que establezca hipotéticamente la identidad de las fijas en el intento (1^{er} parámetro). Cada hipótesis sobre las fijas será ampliada por varias hipótesis que establecerán la identidad de las picas en los valores restantes y sus verdaderas posiciones dentro de la cuádrupla.
Ejemplo: *Permutaciones-P-F* ((1, 2, 3, 4), (1, *?*, *?*, *?*), 2) = {(1, *?*, 2, 3), (1, 3, 2, *?*), (1, 3, *?*, 2), (1, 4, 2, *?*), (1, 4, *?*, 2), (1, *?*, 4, 2), (1, 4, *?*, 3), (1, 3, 4, *?*), (1, *?*, 4, 3)};
El número de hipótesis que genera esta función está acotado como lo muestra la ecuación:

$$\binom{|C|-f}{k} \cdot r_p^{|C|-f-p} \leq |Permutaciones\text{-}P\text{-}F(C, F, k)| \leq \binom{|C|-f}{k} \cdot r_p^{|C|-f}$$

con f el número de fijas en F, o lo que es igual, el número de elementos distinto de *?* en dicha tupla y r las permutaciones ajustadas dada por la fórmula:

$$r_r^n = \begin{cases} 1 & , \text{si } n < r \\ \dfrac{n!}{(n-r)\cdot r!} & , \text{si } n \geq r \end{cases}$$

Ejemplo: *Permutaciones-P-F* ((1, 2, 3, 4), (1, *?*, *?*, *?*), 3) = {(1, 4, 2, 3), (1, 3, 4, 2)}.

Las dos funciones anteriores generan todos los ejemplos positivos que se deducen de los resultados obtenidos tras un intento; cada ejemplo positivo es una hipótesis separada. Es factible generar ejemplos negativos para cada hipótesis; por ejemplo todo valor que no haya sido elegido como una *fija* o una *pica* deberá ser un *desacierto* por lo que no podrá aparecer en ninguna parte del *código*. La función *Desaciertos* recibe el intento original y la hipótesis de los valores y posiciones de las picas y las fijas para retornar todos los posibles casos que rechazan individualmente cada desacierto en cada posible posición.

- Desaciertos(*n-tupla-intento, n-tupla-fijas-picas*) → *Conjunto-de-n-tuplas*

$D \in$ *Desaciertos* $(C, P) \Leftrightarrow (D=(d_1, d_2, ..., d_n)) \wedge (C=(c_1, c_2, ..., c_n)) \wedge (P=(p_1, p_2, ..., p_n)) \wedge \exists q \in N \cap [1, n]$
$\forall i \in N \cap [1, n](i=q \Rightarrow ((d_i \in C) \wedge (d_i \notin P))) \wedge (i \neq q \Rightarrow d_i = *))$

Ejemplo: *Desaciertos*((1, 2, 3, 4), (1, 3, 2, *?*)) = {(4, *, *, *), (*, 4, *, *), (*, *, 4, *), (*, *, *, 4)}.

Debido a la restricción que no permite repetir valores en el juego de *picas&fijas* se pueden generar otros ejemplos negativos que rechacen los *aciertos* en las demás posiciones por ellos ocupadas:

- *No-repeticiones*(*n-tupla-fijas-picas*) → *Conjunto-de-n-tuplas*

$D \in$ *No-repeticiones* $(P) \Leftrightarrow (D=(d_1, d_2, ..., d_n)) \wedge (P=(p_1, p_2, ..., p_n)) \wedge \exists q \in N \cap [1, n] \; \forall i \in N \cap [1, n](i=q \Rightarrow ((d_i \in P) \wedge (d_i \neq p_i))) \wedge (i \neq q \Rightarrow d_i = *))$

Ejemplo: *No-repeticiones* ((1, 2, *?*, *?*)) = { (*, 1, *, *), (*, *, 1, *), (*, *, *, 1), (2, *, *, *), (*, *, 2, *), (*, *, *, 2)}.

3.2 Ejemplos Compuestos Probabilísticos

Cada hipótesis genera varios ejemplos distintos, a lo sumo uno positivo y varios negativos. Sin embargo no es práctico dividir estos ejemplos y asignarles una probabilidad individual; en su lugar debe tratarse el grupo de todos los ejemplos de una hipótesis como un ejemplo compuesto que tiene una probabilidad condicional que cubre al conjunto como un todo pero no a sus elementos: *ejemplo-compuesto* = (*conjunto-ejemplos-simples*, *probabilidad*). El efecto de dicho ejemplo compuesto se calcula actualizando el espacio de versiones de entrada con todos los ejemplos simples y asignándole al resultado final el producto de las probabilidades involucradas.

Por ejemplo si en el juego de Picas & Fijas el intento (1, 2, 3, 4) retorna el resultado *fijas = 1, picas = 1* se pueden generar las siguientes 12 hipótesis cada una con una probabilidad de 1/12: (Fijas = {1}, Picas = {2}), (Fijas = {1}, Picas = {3}), (Fijas = {1}, Picas = {4}), (Fijas = {2}, Picas = {1}), (Fijas = {2}, Picas = {3}), (Fijas = {2}, Picas = {4}), (Fijas = {3}, Picas = {1}), (Fijas = {3}, Picas = {2}), (Fijas = {3}, Picas = {4}, (Fijas = {4}, Picas = {1}), (Fijas = {4}, Picas = {2}), (Fijas = {4}, Picas = {3}).

Si se asume que en el intento (1, 2, 3, 4) el conjunto de fijas es {1} y el de picas es {2}, se pueden generar las siguientes dos hipótesis cada una con una probabilidad condicional de 1/2: que el número 2 va en la 3ª posición ó que va en la 4ª posición.

Si se asume que en el intento (1, 2, 3, 4) el conjunto de fijas es {1}, que el de picas es {2} y que la posición correcta del valor "2" es la 3ª posición, se pueden generar el siguiente concepto compuesto: { ((1, *?*, 2, *?*), *verdadero*), ((3, *, *, *), *falso*), ((*, 3, *, *), *falso*) , ((*, *, 3, *), *falso*), ((*, *, *, 3), *falso*), ((*, 1, *, *), *falso*) , ((*, *, 1, *), *falso*), ((*, *, *, 1), *falso*), ((2, *, *, *), *falso*), ((*, 2, *, *), *falso*) , ((*, *, *, 2), *falso*) } con una probabilidad de 1/24.

3.3 Usando un Espacio de Versiones para Condicionar los Intentos Consecutivos

Lo complicado del juego Master Mind no es calcular todas las hipótesis que se desprenden de un solo resultado individual sino aquellas que se derivan conjuntamente de los resultados de varios intentos consecutivos. Afortunadamente como los resultados de un intento no dependen de los intentos anteriores ni posteriores se puede modelar la incertidumbre de los mismos como si se tratara de un proceso de ruido de la forma en que lo plantearon [6].

El $CEVP_0$ será un conjunto cuyo único elemento tiene al espacio de versiones inicial EV_0 con sus conceptos más generales y más específicos dados por G_0 = {(*, *, *, *)} y E_0 = {(*?*, *?*, *?*, *?*)} y una probabilidad de ocurrencia del 100%:

$$CEVP_0 = CEVP((\phi)) = \{(EV_o, P=100\%)\} = \{((G=(*, *, *, *), E=(?, ?, ?, ?)), P=100\%)\}$$

Cada vez que el adivinador realice un intento los resultados del mismo, la cantidad de picas y fijas, deberán traducirse a ejemplos compuestos probabilísticos que puedan emplearse con el *Algoritmo de Eliminación de Candidatos*. El conocimiento acumulado derivado de los intentos realizados estará almacenado en el *conjunto de espacios de versiones posibles CEVP*. Conforme el juego avance y se realicen los intentos adecuados, algunos de los espacios de versiones reducirán su tamaño pero la mayoría colapsará; estos dos efectos combinados contrarrestarán el número siempre creciente de hipótesis consideradas.

3.3.1 Estrategia de Juego

Siguiendo el planteamiento de [6] el concepto más probable es el mejor candidato a ser el código del oráculo que se debe adivinar. Esta estrategia es directa, fácil de implantar y en general funciona bien, pero tiene un caso patológico donde se desempeña mal si se juega un MaterMind o un Pica&Fijas modificado con más valores posibles en cada posición, por ejemplo con un código en base 16 o superior o con menos de 4 cifras. Supónganse que en un juego de manera muy temprana se han determinado la mayor parte de los números del código y en las posiciones correctas, pero que faltan por concretar los elementos que ocupan unas pocas posiciones en la tupla y la cantidad de candidatos para los mismos es muy alta, dígase q. La estrategia de [6] trataría de adivinar el código de forma directa en cada intento, aún cuando la probabilidad de que esto ocurra es baja, por lo que desperdiciará en promedio $q/2$ intentos antes de acertar el código. Una mejor alternativa sería la de usar las posiciones que ya se conocen con los candidatos por probar con el fin de evaluar y descartar el mayor número de ellos en cada intento.
Ejemplo: El oráculo escoge el número 1239 y rompiendo las reglas da a conocer los tres primeros valores al adivinador. En este punto ya se debe saber que {1, 2, 3} son las fijas, y que el valor faltante debe se uno del conjunto {4, 5, 6, 7, 8, 9, 0} Siguiendo la estrategia del máximo a posteriori se harían los siguientes intentos, no necesariamente en este orden: (1, 2, 3, 4), (1, 2, 3, 5), (1, 2, 3, 6), (1, 2, 3, 7), (1, 2, 3, 8), (1, 2, 3, 9), (1, 2, 3, 0). En promedio necesitarían 3.5 intentos para atinarle al código con un caso peor de 7 intentos.
Una estrategia mejor en estos casos sería la de probar la mitad de los candidatos en cada intento, hasta que el número de los mismos sea menor que el doble del número de posiciones no acertadas.
Intento #1 del adivinador: (4, 5, 6, 7); el oráculo retorna: fijas = 0, picas = 0.
Ahora los candidatos son {8, 9, 0} lo que deja una probabilidad del 33% de ganar en el siguiente intento. Si intento cualquier pareja de este trío de candidatos en la siguiente jugada, se determinará cual de los tres es el valor correcto.
Intento #2 del adivinador: (1, 2, 8, 9); el oráculo retorna: fijas = 3, picas = 0.
Intento #3 del adivinador: (1, 2, 3, 9); el oráculo retorna: fijas = 4, picas = 0.
Fin del juego en 3 jugadas
Incluso si el resultado del 1er intento fueran distinto, se ganaría en un máximo de 3 jugadas: *Fijas = 1, picas = 0* indicaría que el valor faltante es el 7 y se ganaría en dos jugadas; *Fijas = 0, picas = 1* reduciría los candidatos a {4, 5, 6} lo que sería equivalente a la situación contemplada en el intento #2.

4. El algoritmo de Knuth

Knuth presentó un algoritmo capaz de ganar el juego de MasterMind en un máximo de 5 jugadas con un promedio de 4.48 jugadas, [3]. Este algoritmo necesita almacenar de forma explícita el conjunto de todos lo casos posibles para ir tachando tras cada intento aquellos códigos que no son compatibles con los resultados obtenidos. Para generar un nuevo intento, cada intento potencial es comparado con los casos posibles no tachados con el fin de determinar cuantos puede eliminar cada intento en la peor de las suertes. Siguiendo esta estrategia Minimax, se escoge el intento con mayores eliminaciones juzgando cada intento según la mínima cantidad de casos eliminaría. Con pocas modificaciones el algoritmo de Knuth se puede adaptar para jugar el juego de Picas&Fijas.

5. Resultados Experimentales

Se desarrolló un sistema completo de espacio de versiones en CLOS, la extensión orientada a objetos del leguaje LISP, como parte de un proyecto de desarrollo y prueba de un compilador de dicho lenguaje. Sobre esta aplicación se implementaron las estrategias presentadas en este trabajo. También se implemento una versión ampliada del algoritmo de Knuth .

5.1 Resultados en el juego Master Mind.

Knuth	Espacios de Versiones
Jugadas: 500 Numero promedio de Jugadas: 4.482 Desviación estándar número d jugadas: 0.631 Número máximo de Jugadas: 5 Número mínimo de Jugadas: 2	Jugadas: 1000 Numero promedio de Jugadas: 5.516 Desviación estándar número d jugadas: 1.001 Número máximo de Jugadas: 8 Número mínimo de Jugadas: 2

5.2 Resultados en el juego Picas&Fijas.

Knuth	Espacios de Versiones
Jugadas: 200 Numero promedio de Jugadas: 5.33 Desviación estándar número d jugadas: 0.743 Número máximo de Jugadas: 7 Número mínimo de Jugadas: 3	Jugadas: 1000 Numero promedio de Jugadas: 5.27 Desviación estándar número d jugadas: 1.117 Número máximo de Jugadas: 8 Número mínimo de Jugadas: 1

6. Mejorando el Lenguaje de los Conceptos

El lenguaje de las conjunciones de literales positivos y sus negaciones también es compatible con el algoritmo de eliminación de candidatos [8]. El conjunto de conceptos generales en el lenguaje de los literales positivos presenta una explosión combinatoria cuando es especializado mínimamente con miras a rechazar algún ejemplo negativo, producto de la necesidad de expresar literales negativos como la enumeración de todos los demás literales positivos posibles.

Ejemplos: En el juego de Picas&Fijas y el lenguaje L las únicas nueve especializaciones mínimas del concepto ((*,*,*,*), *verdadero*) que rechazan al ejemplo ((1, *, *, *), *falso*) son ((0, *, *, *), *v*), ((2, *, *, *), *v*), ((3, *, *, *), *v*), ((4, *, *, *), *v*), ((5, *, *, *), *v*), ((6, *, *, *), *v*), ((7, *, *, *), *v*), ((8, *, *, *), *v*), ((9, *, *, *), *v*) donde *v* es igual a *verdadero*. El número de especializaciones mínimas del concepto ((*,*,*,*), *verdadero*) que rechazan al ejemplo ((1, 2, *, *), *falso*) son 81 entre las cuales están: ((0, 0, *, *), *v*), ((0, 1, *, *), *v*), ((0, 3, *, *), *v*), ..., ((0, 9, *, *), *v*), ((2, 0, *, *), *v*), ..., ((9, 7, *, *), *v*), ((9, 8, *, *), *v*) aunque algunos conceptos (ocho) pueden se anulados por que contienen valores repetidos. El número de especializaciones mínimas del concepto ((*,*,*,*), *verdadero*) que rechazan al ejemplo ((1, 2, 3, 4), *falso*) son 6561 de los que se pueden anular algunos.

Una buena alternativa es usar un lenguaje que permita describir fácilmente cuales han sido los elementos descartados ya que el número de desaciertos supera ampliamente al de aciertos pues de todos lo códigos posibles solo uno es el correcto y el 60% de los valores son desaciertos en la Picas&Fijas y hasta un 90% en MasterMind. En cada posición de la cuádrupla se deben poder especificar varios valores descartados, contrario al único valor correcto que puede existir en cada posición. El lenguaje de las conjunciones de las negaciones de disyunciones por cada posición de la cuádrupla cumple con todos estos requisitos.

El operador *Ni* (*nor*) también conocido como la flecha de Pierce es equivalente a la negación de disyunciones: $(x_1 \downarrow x_2 \downarrow ... \downarrow x_n) \equiv \neg(x_1 \vee x_2 \vee ... \vee x_n)$. En el nuevo lenguaje cualquier grupo de literales válidos unidos por operadores *ni* es un valor posible para cada posición de la cuadrupla. Generalizando para una n-tupla:

$$L_2=\{\ c : c=(m_1, m_2, ... ,m_n) \wedge \forall i \in N\ (1 \le i \le n) \Rightarrow\ (\ m_i \in \{\neg *, \neg ?\} \vee (\ (k \ge 1) \wedge (m_i = \neg\ (x_{i1} \vee x_{i2},\ \vee ... \vee x_{ik}) \wedge \forall j \in N\ (1 \le i \le k) \Rightarrow (x_{ij} \in D_i)\ )\ )\ \}$$

En L_2 los conjuntos iniciales son G_o = {((¬*,¬*,¬*,¬*), *verdadero*)} y E_o = {((¬?,¬?,¬?,¬?), *verdadero*)}. ¬* se puede interpretar como "ningún literal" mientras que ¬? se puede interpretar como "cualquiera de los literales". También se tiene que (¬* = ?) y (¬? = *).

Ejemplo: En L_2 el número de especializaciones mínimas del concepto ((¬*,¬*,¬*,¬*), *verdadero*) que rechazan al ejemplo ((¬1, *, *, *), *falso*) es uno: ((0 ↓ 2 ↓ 3 ↓ 4 ↓ 5 ↓ 6 ↓ 7 ↓ 8 ↓ 9,), *verdadero*); este caso equivale a confirma que 1 es una fija en la primera posición.

7. Trabajo Futuro

Una de las primera aplicaciones basada en espacios de versiones fue LEX que efectuaba integración simbólica empleando los espacios para la generación de heurísticas que permitieran determinar el orden en que las diferentes técnicas de integración tradicionales debían aplicarse a cada tipo de formula para poder hallar exitosamente su antiderivada.

Un buen compilador optimizador contará con un repertorio de optimizaciones posibles sobre los códigos fuente y generado, pero el desempeño final dependerá del orden que se apliquen dichas optimizaciones. Un planteamiento similar al empleado en LEX permitiría el desarrollo de heurísticas que indiquen en que orden aplicar técnicas de optimización conocidas a diferentes patrones de código de forma que se maximice el rendimiento final.

Reconocimiento

Este trabajo inició como una parte de una tesis de Maestría en Sistemas dirigida por el profesor Luis Roberto Ojeda en la que se está desarrollando un compilador de LISP. Fue él quién propuso el tema de los Espacios de Versiones para ser desarrollado en el lenguaje y adquirir así un buen manejo del mismo. La aplicación resultante servirá para comprobar el funcionamiento del compilador una vez terminado.

Referencias

[1] Cohen, Paul R. and Feigenbaum, Edward A.. *The Handbook of Artificial Intelligence*, Volumen 3. HeurisTech Press - William Kaufman, Inc. 1982, pages 383-419, 484-493.

[2] Hirsh, Haym. *Polynomial-Time Learning with Version Spaces. AAAI 1992, pages 117-122.*

[3] Knuth, Donald E. *The Computer as a Master Mind, Journal of Recreational Mathematics (9)*: 1-6 (1976-77).

[4] Luger, George F. *Artificial Intelligence - Structures and Strategies for Complex Problem Solving*, 4ª edition. Addison Wesley, 2002, pages 351-372.

[5] Mitchell, Tom M. *Machine Learning,* 1st edition, pages 20-45. McGraw-Hill (March 1), 1997.

[6] Norton, Steven W. and Hirsh, Haym. *Classifier Learning from Noisy Data as Probabilistic Evidence Combination. ICML - International Conference on Machine Learning* - 1993, *pages* 220-227.

[7] Norton, Steven W. and Hirsh, Haym. *Learning DNF Via Probabilistic Evidence Combination. AAAI 1992, pages 141-146*.
[8] Poole, David Mackworth, Alan and Goebel, Randy. *Computational Intelligence - A Logical Approach*. Oxford University Press, USA, (January 8), 1998, pages 416-423.
[9] Rich, Elaine and Knight, Kevin, *Inteligencia Artificial*, 2nd edition. McGraw-Hill. 1994 traducido de la 2ª edition en inglés de *Artificial Intelligence*, 1991, pages 510-517.