Revista Cubana de Ciencias Informáticas
Vol. 11, No. 2, Abril-Junio, 2017
ISSN: 2227-1899 | RNPS: 2301
http://rcci.uci.cu
Pág. 29-45

Tipo de artículo: Artículo original
Temática: Inteligencia Artificial
Recibido: 07/10/2016 | Aceptado: 24/02/2017

Modelo de agente lógico con inferencia basada en hechos

Model of logic agent with facts-based inference

Yuniesky Coca Bergolla [1*], Leduan Bárbaro Rosell Acosta [2], Arletis Velázquez Ramírez [2]

[1] Dirección docente metodológica, Universidad de las Ciencias Informáticas. Carretera a San Antonio de los Baños km 2 ½ Reparto Torrens. La Habana. ycoca@uci.cu

[2] Departamento de Programación y Sistemas Digitales, Facultad 4, Universidad de las Ciencias Informáticas. Carretera a San Antonio de los Baños km 2 ½ Reparto Torrens. La Habana. {leduanb, arlette}@uci.cu

[*] Autor para correspondencia: ycoca@uci.cu

Resumen

La programación lógica se ha mantenido como uno de los principales paradigmas dentro de las investigaciones en el área de la inteligencia artificial. Por otro lado, la inferencia basada en casos es muy utilizada para la creación de sistemas expertos, sobre todo en problemas de clasificación o diagnóstico, donde se puede determinar fácilmente un objetivo o variable dependiente y un conjunto de atributos condicionalmente independientes entre sí, dado el objetivo. En el presente trabajo se utilizan las potencialidades declarativas de la programación lógica y las ventajas del razonamiento basado en casos para elaborar un modelo de agente lógico con inferencia basada en hechos. El mismo fue implementado mediante el desarrollo de una herramienta de código abierto para la enseñanza de la Inteligencia Artificial en la Universidad de las Ciencias Informáticas. El modelo de agente propuesto determina, a partir de sus creencias, el peso de cada atributo aplicando elementos de la teoría de la información, el cálculo de la incertidumbre mediante la teoría de las probabilidades y siguiendo un enfoque de Bayes Ingenuo, y brinda información útil para la toma de decisiones en la incorporación de nuevos casos a la base de conocimiento. Además, realiza el aprendizaje mediante la actualización de los valores de creencia de cada valor al realizar nuevas inferencias y al incorporar nuevos casos a la base. El modelo fue implementado y probado, alcanzando resultados de rendimiento favorables.

Palabras clave: Agente lógico, Prolog, Razonamiento basado en casos.

Abstract

Logic programming has remained as one of the main paradigms in research in the area of artificial intelligence. Furthermore the case-based inference is very used for creating expert systems, especially in problems about classification or diagnosis, which can easily determine a target or dependent variable and a set of conditionally

Revista Cubana de Ciencias Informáticas
Vol. 11, No. 2, Abril-Junio, 2017
ISSN: 2227-1899 | RNPS: 2301
http://rcci.uci.cu
Pág. 29-45

independent attributes each other, given the objective. In this paper is proposed a logical agent model with inference based on facts, using the declarative potential of logic programming and the advantages of the case-based reasoning. The model was implemented by developing an open source tool for teaching Artificial Intelligence at the University of Informatics Science. The proposed agent determines, based on their beliefs, the weight of each attribute using elements of information theory, also calculates uncertainty following a Naive Bayes approach, and provides useful information for decision-making in the incorporation of new cases to the knowledgebase. It also performs learning by updating the belief values of each value to make new inferences and incorporate new cases to the base. The model was implemented and tested, the performance results achieved were favorable.
Keywords: *Case-based reasoning, logic agent, Prolog.*

Introducción

El poder expresivo de la lógica de primer orden (LPO) le ha permitido ser tomada muy en serio por los investigadores en el área de la inteligencia artificial. La inferencia hacia atrás en la LPO permitió la aparición de la programación lógica, esta representa la base de Prolog (Bratko, 1990), uno de los lenguajes declarativos más reconocidos y utilizado todavía hoy. A pesar de ser visto por algunos como poco práctico por sus limitantes en eficiencia computacional, se han llevado a cabo investigaciones, buscando mejoras a sus compiladores. Estas han permitido la obtención de resultados comparables con lenguajes reconocidos por su eficiencia como el C++ (Russell y Norving, 2010a).

Algunos de los sistemas expertos más recientes desarrollados con Prolog son los referenciados en (Atanasova and Krupka, 2013) (Zhang, et al. 2011) (Kerdprasop and Kerdprasop, 2011), (Kadhim, et al. 2011). Pero las investigaciones no se han limitado a construir sistemas basados en este lenguaje, también se han dedicado esfuerzos para facilitar el trabajo de los desarrolladores, sobre todo a través del vínculo con lenguajes imperativos. JPL (Ali, et al. 2016) es una biblioteca que permite integrar de forma fácil y eficiente el Swi-Prolog con Java, C++ y otros lenguajes imperativos, lo que hace mucho más atractiva la utilización de este lenguaje declarativo en la construcción de sistemas expertos. En cuanto a arquitecturas, Guoqi presenta un patrón de diseño basado en la comunicación entre procesos para aprovechar la inferencia de Prolog con aplicaciones o sistemas embebidos (Guoqi, et al. 2014). También, Dunstan propone una arquitectura híbrida para sistemas expertos basados en web, la cual convierte información XML en código Prolog para realizar inferencia lógica (Dunstan, 2012). Otras investigaciones han aprovechado las potencialidades de Prolog para mejorar algoritmos clásicos como las búsquedas (Ali, et al. 2014); en dicho trabajo se presenta un sistema de revisión de creencias que permite no volver a evaluar nodos que son previamente visitados.

Por otra parte, la programación lógica ha evolucionado, lo cual ha permitido el desarrollo de extensiones del Prolog como JavaLog (Zunino, et al. 2001) para la creación de agentes lógicos; el ProbLog (de Raet, 2015) para crear programas de lógica probabilística, el B-prolog (Zhou, 2012) que combina los paradigmas de programación lógica y basada en restricciones. Estos permiten ampliar significativamente el área de impacto de la programación lógica dentro de la Inteligencia Artificial (IA).

La IA se ha definido de varias formas, sin embargo, la más aceptada en nuestros días la asume como la rama de las ciencias de la computación encargada de la construcción de sistemas que actúen racionalmente. Esta idea ha llevado al desarrollo de un nuevo paradigma: la programación basada en agentes. Se puede definir el término agente como un sistema que percibe información y actúa de forma autónoma sobre un entorno específico. Por tanto, el objetivo de la IA es diseñar un programa agente que implemente una función que transforma un conjunto de percepciones en acciones. Estas acciones del agente definen su comportamiento y a través de este modifica el entorno. La arquitectura interna del agente puede tener varias formas, sin embargo se tiene consenso en cuanto a las funciones básicas de percibir el entorno y accionar en el mismo. El modelo BDI (creencias, deseos e intenciones por sus siglas en inglés) es uno de los más difundidos y aceptados por la comunidad científica (Henderson-Sellers, et al. 2005). Según este modelo, el agente puede ser representado como un conjunto de creencias, asumidas luego de percibir el entorno; un conjunto de intenciones, establecidas a partir de sus acciones; y uno o varios deseos que definen el objetivo del agente y le permite comportarse de manera autónoma en el entorno.

El entorno determina en gran medida el diseño interno del agente, si es completamente observable no se necesita hacer un seguimiento, o sea, tener una memoria. Como generalmente los entornos no son completamente observables, entonces, las acciones del agente dependen de una secuencia de percepciones que requieren tener algún tipo de memoria. Russell y Norving (2010b) proponen una tipología de agentes en base a su complejidad interna. Según estos autores los agentes reflexivos simples responden directamente a la percepción del entorno; los agentes reflexivos basados en modelos mantienen una memoria que permite dar seguimiento a elementos del entorno no evidentes en la percepción de cada instante; los agentes basados en metas analizan los posibles estados a los cuales llegarían, si seleccionan determinada acción; y los agentes basados en utilidad tratan de maximizar su utilidad, a partir de su rendimiento en el entorno. Todos ellos pueden ser agentes aprendices, en la medida que mejoren su comportamiento en el entorno.

En cuanto al entorno estos autores refieren varias formas de representar lo que el agente hace, en tanto definen como tipos de estados del entorno:

- **Atómicos:** Se asume el estado como indivisible, sin estructura interna.

- **Factorizados:** Asume los estados como un conjunto de pares variable-valor. Esta representación permite el análisis de la incertidumbre.
- **Estructurados:** Los estados son representados por objetos y sus relaciones. Es utilizada en las más avanzadas técnicas de Inteligencia Artificial, incluida la LPO.

Precisamente la LPO, y más específicamente la programación lógica, es muy utilizada para la construcción de sistemas expertos por su forma declarativa de representar el conocimiento. Generalmente se realiza la representación en forma de reglas, lo cual exige de expertos en el dominio para definirlas. Por su parte, los sistemas de aprendizaje basados en casos son deseables en muchas ocasiones, pues solo reciben hechos del mundo y a partir de ellos el sistema o agente resuelve nuevos problemas.

En la bibliografía consultada no se ha encontrado una arquitectura general para el diseño de un agente lógico que utilice una base de ejemplos para realizar la inferencia con un análisis autónomo de la incertidumbre. Un agente con estas características sería de interés para la enseñanza de temas de la Inteligencia Artificial, como el lenguaje Prolog, los sistemas basados en conocimiento y el tratamiento de la incertidumbre. El presente trabajo se traza como objetivo proponer un modelo de agente aprendiz basado en utilidad para entornos con estados estructurados en forma de predicados de lógica de primer orden, específicamente hechos definidos en lenguaje Prolog. El agente realiza la inferencia lógica basada en hechos, asigna a partir de sus creencias los pesos a cada rasgo, utilizando el análisis de ganancia de información y un análisis probabilístico, basado en Bayes Ingenuo, para el tratamiento de la incertidumbre. Se realiza el aprendizaje mediante la actualización de los valores de creencia del agente, tanto a la hora de incorporar nuevos casos a la base de conocimiento, como en cada nueva inferencia realizada. El modelo propuesto se implementa mediante el desarrollo de un sistema para la enseñanza y aprendizaje de la Inteligencia Artificial en la Universidad de las Ciencias Informáticas.

Metodología computacional

El agente se diseñó siguiendo el modelo BDI. Las creencias del agente se definieron a partir de las percepciones del entorno y la base de conocimiento interna. Se asumió como deseo del agente la solución de un nuevo caso y se definieron las intenciones a partir de cada una de las acciones que lleva a cabo, tanto de cara al usuario como internamente.

Definición del agente lógico basado en ejemplos

Revista Cubana de Ciencias Informáticas
Vol. 11, No. 2, Abril-Junio, 2017
ISSN: 2227-1899 | RNPS: 2301
http://rcci.uci.cu
Pág. 29-45

Un agente lógico debe contar con una base de conocimiento que guarde sus creencias. Estas se pueden representar como un conjunto de expresiones arbitrarias, conocidas como cláusulas. Cada cláusula puede expresarse de la forma:

$$q: -p_1, p_2, \dots, p_n \qquad (1)$$

Esta sentencia expresa que q será verdadera si cada una de las $p_j (j = 1, \dots, n)$ son verdaderas. Las expresiones a la izquierda y la derecha del operador $: -$ deben ser átomos o expresiones de la forma:

$$h(\, t_1, t_2, t_3, \dots, t_m) \qquad (2)$$

Donde h es una relación constante y cada $t_i (i = 1,2, \dots, m)$ es un término. Un término puede ser un objeto constante, una variable o una estructura de la forma:

$$e(\, s_1, s_2, s_3, \dots, s_q) \qquad (3)$$

Donde *e* es una función constante y cada $s_k (k = 1,2, \dots, q)$ es un término. Existen cláusulas que no tienen parte derecha y son siempre verdaderas, son llamadas hechos y tienen la forma:

$$q. \qquad (4)$$

Un ejemplo *E* se define como un par (X, f(X)), donde X es un vector de entrada, y f(X) es la salida de la función f aplicada al vector X. En problemas de clasificación, diagnóstico y muchos otros, X representa el conjunto de rasgos predictores o independientes (atributos en lo adelante) y f(X) el rasgo dependiente (objetivo en lo adelante). Para la construcción de árboles de decisión a partir de ejemplos (Mitchel, 1997) se calcula la ganancia de información de cada atributo de X hacia f(X), pero no se asume una interdependencia entre ellos. Por otro lado el modelo de Bayes Ingenuo (Russell y Norving, 2010c) asume que los atributos son condicionalmente independientes entre sí, dado el objetivo. A partir de estos elementos en lo sucesivo se asume que el agente guardará ejemplos y resolverá problemas con atributos de X condicionalmente independientes entre sí, dada f(X). Para simplificar la exposición, se asumen los atributos discretizados. En caso de necesitarse, el agente puede realizarlo de forma autónoma mediante un algoritmo como CAIM (Kurgan and Cios, 2004) o Ameva (González-Abril, et al. 2009).

Para facilitar los cálculos y tomando en cuenta la representación recursiva que hace la programación lógica de las listas, se propone una estructura para los ejemplos de la forma:

$$ejemplo(id, rasgo(valor_0,\ cree_0), rasgo(valor_1,\ cree_1), \dots, rasgo(valor_m,\ cree_m)) \qquad (5)$$

Donde el primer rasgo es el objetivo y los siguientes son los atributos.

Revista Cubana de Ciencias Informáticas
Vol. 11, No. 2, Abril-Junio, 2017
ISSN: 2227-1899 | RNPS: 2301
http://rcci.uci.cu
Pág. 29-45

Una representación usual para este tipo de sistema es a través de una matriz donde cada fila representa un ejemplo y cada columna un rasgo, se asume que la primera columna es el objetivo.

Tabla 1. Representación de una base de casos

	r_o	. . .	r_i	. . .	r_m
e_1	$v(r_o e_1)$	. . .	$v(r_i e_1)$	. . .	$v(r_m e_1)$
.					
.					
.					
e_j	$v(r_o e_j)$	. . .	$v(r_i e_j)$	. . .	$v(r_m e_j)$
.					
.					
.					
e_n	$v(r_o e_n)$	. . .	$v(r_i e_n)$	. . .	$v(r_m e_n)$

Donde cada $v(r_i e_j)$ representa un valor del conjunto de valores posibles del atributo r_i y cada $v(r_o)$ representa un valor del conjunto de valores posibles del objetivo r_o.

Algunos elementos necesarios para el trabajo posterior del agente será guardar la información necesaria acerca de cada uno de los rasgos, tanto los atributos como el objetivo. La representación de cada rasgo debe respetar el orden en que aparecerán en la estructura *ejemplo* creada, ya que la programación lógica realiza la inferencia siguiendo un orden de primero en profundidad. Los hechos tendrán la forma siguiente:

$$rasgo(i, r_i, t, D, w_i).$$

Donde:

i: Número consecutivo para identificar la posición de cada rasgo en el ejemplo.

r_i: Nombre del rasgo.

t: Tipo de dato, el cual puede ser nominal, real o entero.

D: Dominio. Para el caso de los nominales tiene un conjunto de valores posibles, aquí se incluyen los dicotómicos. Los numéricos tienen el mínimo y el máximo valor.

w_i: Peso del rasgo.

Cálculo de los pesos

En el razonamiento basado en casos los atributos no aportan de la misma manera al objetivo. En muchas ocasiones se asignan pesos a los atributos a partir del criterio de expertos. Sin embargo, un agente debe ser capaz de comportarse de forma autónoma, sin depender de la intervención del usuario. El uso de la teoría de la información para el cálculo de la

Revista Cubana de Ciencias Informáticas
Vol. 11, No. 2, Abril-Junio, 2017
ISSN: 2227-1899 | RNPS: 2301
http://rcci.uci.cu
Pág. 29-45

entropía, definida por Shanon y Weaver en 1949 (DMITRIEV, 1991) utilizada en la construcción de árboles de decisión, puede utilizarse para asignar un peso o importancia a cada atributo a partir de la información que tiene el agente, es decir, a partir de sus creencias.

La teoría de la información mide el contenido de información en bits. Un bit de información es suficiente para responder una pregunta de tipo sí/no sobre la que no se sabe nada. En general, si las respuestas posibles v_i tienen probabilidades $P(v_i)$, el contenido de información *I* de la respuesta actual viene dado por:

$$I(P(v_1),\ldots,P(v_k)) = \sum_{i=1}^{k} -P(v_i)\ log\ _2 P(v_i) \qquad (6)$$

Donde *k* es la cantidad de respuestas posibles. Como $P(v_i)$ toma valores entre 0 y 1, el valor resultante de información está acotado entre 0 y log_2k.

En los sistemas basados en casos la tarea más común es saber cuál es el valor del objetivo en un nuevo problema *nc*. De ahí que se calcule la información *I* para el objetivo r_o en los ejemplos de la base de conocimiento *C*. Luego cualquier atributo r_i divide el conjunto de ejemplos *C* en subconjuntos $C_1,\ldots C_p$ de acuerdo con sus valores para r_i, donde r_i puede tener *p* valores distintos. Para cada subconjunto C_j se calcula nuevamente la necesidad de Información *I* del objetivo r_o. En general para calcular la información a partir del análisis del atributo r_i se tiene:

$$Resto(r_i) = \sum_{j=1}^{p} \frac{n(C_j)}{n(C)}\ I(P'(v_1),\ldots,P'(v_k)) \qquad (7)$$

Donde $n(C_j)$ es la cardinalidad del subconjunto C_j, *n(C)* es la cardinalidad del conjunto de ejemplos *C* y $P'(v_1),\ldots,P'(v_k)$ es la probabilidad de cada respuesta del rasgo objetivo r_0 dentro del subconjunto C_j.

La ganancia de información del atributo r_i es la diferencia entre la necesidad de información original (7) y la nueva necesidad de información (8):

$$Ganancia(r_i) = I(P(v_1),\ldots,P(v_k)) - Resto(r_i) \qquad (8)$$

El inconveniente fundamental de este cálculo es cuando los valores posibles de un atributo son muchos, en ocasiones pueden llegar a ser diferentes para cada caso, ejemplo los atributos identificadores. En esos casos el resultado de *(9)* asignaría el mayor peso a dicho atributo, cuando realmente no aporta al valor del objetivo. Para lograr un mayor aporte de los atributos con similar cantidad de valores posibles al objetivo, se divide el valor obtenido entre el módulo de la diferencia de los valores posibles del atributo (r_i) y el rasgo objetivo (r_0):

$$GananciaM(r_i) = \frac{Ganancia(r_i)}{1+|p-k|} \qquad (9)$$

Revista Cubana de Ciencias Informáticas
Vol. 11, No. 2, Abril-Junio, 2017
ISSN: 2227-1899 | RNPS: 2301
http://rcci.uci.cu
Pág. 29-45

Donde p es la cantidad de valores posibles para el atributo (r_i) y k es la cantidad de valores posibles para el rasgo objetivo r_0. Este resultado es mucho más aceptable, pero aún los rasgos identificadores tendrán valores superiores a otros atributos. Para eliminar cualquier posibilidad de estar analizando un rasgo identificador, se adiciona una regla dura de manera que asigne:

$$Ganancia(r_i) = 0 \ Si \ \ p = n(C)$$

Es decir que la Ganancia del atributo será 0 si la cantidad de valores posibles p es igual a la cantidad de ejemplos.

Con este valor y para garantizar que $\sum_{i=1}^{m} w_i = 1$, se calcula la sumatoria de las ganancias de los atributos

$$G = \sum_{i=1}^{m} GananciaM(r_i)$$

y a partir de este valor se calcula el peso de cada atributo: $w_i = \frac{GananciaM(r_i)}{G}$

Tratamiento de la incertidumbre

En los sistemas de razonamiento basados en casos pueden estar presente diversas fuentes de incertidumbre. El agente puede determinar un grado de creencia de cada valor de los atributos, a partir de los ejemplos guardados en su base de conocimiento. A pesar de ser cuestionada, debido al uso simplificado de las hipótesis de independencia, la teoría de la probabilidad sigue siendo la más aceptada para el tratamiento de la incertidumbre (Russell y Norving, 2010c). Soporta este criterio el hecho de ser la menos dependiente del factor humano y brindar facilidades para el cálculo autónomo por el agente, lo cual la hace superior a otras teorías como los factores de certeza, la credibilidad, la plausibilidad, necesidad o posibilidad. Para la presente propuesta se seguirá un análisis similar al modelo Bayes Ingenuo, para lo cual se calcula el nivel de creencia del agente sobre cada valor de los atributos, tomando en cuenta dos magnitudes:

- $Cree_p$ Creencia calculada a partir de la probabilidad a priori de la existencia de dicho valor para ese atributo en la base de conocimiento.
- $Cree_c$ Creencia que toma en cuenta la relación de dicho valor para ese atributo con el valor del rasgo objetivo en ese mismo caso.

Para cada uno de los valores de los atributos en cada ejemplo de la base de conocimiento se define formalmente cada una de las creencias como:

$$Cree_p(v(r_i)) = P(v(r_i)) \qquad (10)$$

$$Cree_c(v(r_i)) = P(v(r_i) \mid v(r_0)) \qquad (11)$$

Revista Cubana de Ciencias Informáticas
Vol. 11, No. 2, Abril-Junio, 2017
ISSN: 2227-1899 | RNPS: 2301
http://rcci.uci.cu
Pág. 29-45

Para calcular la creencia total de cada v_i se utiliza una función para combinarlas, se asume que cada una aporta positivamente a la creencia total, tanto una alta probabilidad de ocurrencia de un valor $Cree_p$, como una alta probabilidad de que ese valor aporte a una conclusión $Cree_c$, brinda elementos para creer que el valor tiene menos incertidumbre:

$$Sum(x,y) - Prod(x,y) = x + y - x * y \qquad (12)$$

De esta forma $Cree_t = Cree_p + Cree_c - Cree_p * Cree_c$

Luego se debe obtener un valor de creencia para el rasgo objetivo, dada la creencia de cada uno de los valores de los rasgos predictores:

$$Cree_t(v(r_0)|v(r_1)\ldots,v(r_m)) = P(v(r_0)) \prod_i Cree_t(v(r_i))$$

La creencia del valor del atributo r_i ya incluye en su cálculo la probabilidad condicional $P(v(r_i) \mid v(r_o))$.

Recuperar los casos

Un algoritmo general para recuperar los casos semejantes se propone en (Coca, 2003), donde se utiliza una medida δ_i. Esta brinda un valor de semejanza en el rango [0,1] entre los valores del atributo r_i del caso nuevo *nc* y el caso con que se compara en la base de conocimiento e_j. Algunas de las medidas δ más utilizadas son:

Para atributos con valores dicotómicos:

$$\delta_i = \begin{cases} 1 & si\ v(r_i e_j) = v(r_i nc) \\ 0 & e.o.c \end{cases} \qquad (13)$$

Para atributos con valores numéricos:

$$\delta_i = 1 - \left| \frac{v(r_i e_j) - v(r_i nc)}{v_{max}(r_i) - v_{min}(r_i)} \right| \qquad (14)$$

En muchas ocasiones, sobre todo tomando como base la teoría de los conjuntos borrosos, los valores numéricos son discretizados asumiendo valores nominales, para los cuales se definen funciones de semejanza más específicas. Asumiendo que la vecindad entre los conjuntos determina la semejanza entre ellos, se pueden asignar valores numéricos consecutivos a los conjuntos, lo cual permite aplicar la función *(15)*.

Además, se requiere de una función de semejanza *f* para combinar los valores de las semejanzas entre los atributos ponderados por el peso.

$$f\left(w_i, \delta_i\left(v(r_i e_j), v(r_i nc)\right)\right) = \frac{\sum_{i=1}^{m} w_i * \delta_i\left(v(r_i e_j), v(r_i nc)\right)}{\sum_{i=1}^{m} w_i} \qquad (15)$$

Revista Cubana de Ciencias Informáticas
Vol. 11, No. 2, Abril-Junio, 2017
ISSN: 2227-1899 | RNPS: 2301
http://rcci.uci.cu
Pág. 29-45

Algunos autores incluyen en la función de semejanza entre casos el tratamiento de la incertidumbre para obtener un valor de semejanza β que incluye en su resultado dicho valor. Sin embargo en la presente propuesta el tratamiento de la incertidumbre se hace aparte del valor de semejanza calculado. Se realiza toda la inferencia de forma paralela, esto permite brindar un valor de semejanza en base a las creencias del agente y un valor de creencia que tiene el propio agente sobre el resultado obtenido. De esta forma la salida del agente incluirá para cada caso semejante un par (Valor, Creencia). Quedando el algoritmo específico utilizado de la siguiente manera:

Entrada: *nc*, *ε, C* (nuevo problema, valor mínimo de semejanza para recuperar los ejemplos y conjunto de ejemplos de la base)

Salida: $L_s = \left\{\left(\mathrm{e'}_1, \mathrm{Par}\left(\beta(\mathrm{nc}, \mathrm{e'}_1), \mathrm{Cree}_t(\beta)\right)\right), \ldots, \left(\mathrm{e'}_s, \mathrm{Par}\left(\beta(\mathrm{nc}, \mathrm{e'}_s), \mathrm{Cree}_t(\beta)\right)\right)\right\}$ (Lista formada por cada ejemplo semejante y el par formado por la medida de semejanza entre *nc* y cada e', y la creencia total del agente respecto al valor obtenido)

Asignar $v(r_o cn) = v(r_o e_j)$

Para cada atributo r_i de *nc*, así como para el objetivo, calcular $Cree_p$ y $Cree_c$

Para cada ejemplo e_j de la base de conocimiento *C* hacer:

Para cada atributo r_i de e_j hacer:

Se buscan los valores $v(r_i e_j)$ y $v(r_i nc)$ (valores del atributo r_i en *nc* y e_j)

Calcular la semejanza $\delta_i\left(v(r_i e_j), v(r_i nc)\right)$ entre dichos valores

Calcular la semejanza $\beta(nc, e_j)$ entre el ejemplo e_j y el nuevo problema *nc* a través de la función:

$$\mathrm{f}(\mathrm{e}_j, \mathrm{nc}) = \frac{\sum_{i=1}^{m} w_i * \delta_i\left(v(r_i e_j), v(r_i nc)\right)}{\sum_{i=1}^{m} w_i}$$

Calcular la creencia conjunta de los valores de los rasgos objetivos de *nc* y e_j mediante la regla del producto de la teoría de las probabilidades:

$$CreeC(nc, e_j) = Cree(r_o cn) * Cree(r_o e_j)$$

Calcular la creencia del valor obtenido en $\beta(nc, e_j)$ a través de la función θ:

$$\theta(nc, e_j) = 1 - \left|\beta(nc, e_j) - \mathrm{CreeC}(nc, e_j)\right|$$

Comparar el valor ε con $\beta(nc, e_j)$

Si $\beta(nc, e_j) > \varepsilon$ adicionar a la lista el ejemplo, el valor de semejanza con el nuevo problema y la creencia del agente respecto a dicho valor:

$$L_s \leftarrow \left\{e_j, \mathrm{Par}(\beta(nc, e_j), \theta(nc, e_j))\right\}$$

Revista Cubana de Ciencias Informáticas
Vol. 11, No. 2, Abril-Junio, 2017
ISSN: 2227-1899 | RNPS: 2301
http://rcci.uci.cu
Pág. 29-45

Adaptar la solución

La adaptación en los SBC puede hacerse compleja en problemas de diseño y planificación u otros similares, sin embargo en el tipo de problemas numéricos que se presenta, la adaptación se limita a encontrar una solución final al problema, a partir de los resultados obtenidos en los casos recuperados. Las vías para encontrar la solución definitiva son diversas, pero en todo caso dependerá del dominio del objetivo r_o. Este cálculo no se realiza de la misma manera para valores nominales, numéricos enteros o numéricos reales. Varios autores incluyen en el resultado final el análisis de los valores de incertidumbre, en la presente propuesta se realiza el cálculo del valor final y la creencia que tiene el agente sobre dicho resultado de forma separada. La lista de casos recuperados L_s debe ser considerada por el agente para obtener una propuesta de solución al problema. Esta solución será dependiente del tipo de datos que guarde el objetivo r_o:

- **Rasgo nominal:** Se utiliza la moda, es decir, el valor del objetivo r_o que más se repite dentro de la lista L_s. Para obtener el valor de creencia se combinan los valores de cada caso que tiene por valor objetivo la moda, según la fórmula *(13)*.
- **Rasgo numérico real:** Se calcula el promedio de los valores de los objetivos r_o de toda la lista L_s. Como los valores de creencia están asociados a valores que serán promediados, el nuevo valor de creencia se calculará de la misma manera, o sea, calculando el promedio de las creencias de cada valor.
- **Rasgo numérico entero:** Se calcula el promedio truncado para el valor final, mientras el valor de creencia se calcula similar a los rasgos numéricos reales.

Aprendizaje y explicación

El aprendizaje más común en este tipo de sistemas se logra mediante la incorporación de nuevos casos a la base de conocimiento. Este proceso puede hacerse de forma automática, ya sea directamente adicionando cada nuevo caso resuelto o analizando si tiene elementos que aporten de forma significativa a nuevos posibles casos a resolver. Un agente debe ser capaz de decidir cuándo un caso aporta significativamente al sistema, o al menos tener elementos para tomar la decisión. Algunos de los más significativos pueden ser:

- **Tamaño n de la base de conocimiento:** Mientras mayor sea, el sistema debe ser más restrictivo en la adición de nuevos casos.
- **Cantidad de casos semejantes recuperados (longitud de L_s):** Si es baja, es muy probable que el nuevo caso tenga elementos de interés en la solución de futuros problemas.
- **Valor de semejanza entre el nuevo caso y el caso más semejante de la base ($\beta(nc, e_j)$*):** Si este valor es bajo se puede asumir que el nuevo caso es significativamente diferente a los guardados.

Revista Cubana de Ciencias Informáticas
Vol. 11, No. 2, Abril-Junio, 2017
ISSN: 2227-1899 | RNPS: 2301
http://rcci.uci.cu
Pág. 29-45

- **Valor de creencia obtenido del rasgo objetivo del nuevo caso (Cree($r_o$nc)):** Si se obtiene un valor bajo puede que la solución no sea tan buena, por tanto su incorporación a la base pudiera afectar en alguna medida las soluciones posteriores.

Se pueden tomar en cuenta otros elementos, pero encontrar una función donde el agente pueda tomar la decisión correcta, requiere aplicar algún método de optimización que complejizaría el proceso, lo cual no es objetivo en este trabajo. Por otro lado una de las ventajas de los sistemas basados en casos está en la posibilidad de tener un módulo de explicación, donde se ofrecen al usuario los elementos fundamentales que permitieron al sistema llegar a un resultado. A partir de lo anterior se propone brindar al usuario los elementos expuestos con anterioridad, de esta forma puede determinar la incorporación o no del nuevo caso a la base de conocimiento.

El aprendizaje del sistema no termina con la incorporación del caso, a partir de ahí se deben actualizar las creencias del agente. Además, con la solución de cada nuevo caso el sistema puede aprender de los valores que trae cada atributo. A partir de lo cual se asume el aprendizaje del agente en dos momentos:

1. **Al realizar una inferencia:** Se actualizan los valores de $Cree_p$. Se asume el valor de cada atributo del nuevo caso $v(r_i nc)$ como información para la actualización de dicho valor de creencia a priori de cada ejemplo de la base.
2. **Al incorporar nuevo ejemplo a la base:** Se actualizan los valores de creencia $Cree_c$ y $Cree_t$, para este último se utiliza el ya actualizado $Cree_p$.

Resultados y discusión

El modelo de agente lógico con inferencia basada en hechos se implementó en el desarrollo de un sistema inteligente para la resolución práctica de problemas en la disciplina Inteligencia Artificial de la Universidad de las Ciencias Informáticas. Para el desarrollo del mismo se utilizó la plataforma de desarrollo Java FX 2.2 con Netbeans 8.0 y el SWI-Prolog 6.2.6 con JPL como enlace entre Prolog y Java. Se propuso una arquitectura basada en el modelo BDI (Figura 1), dejando bien definida la interfaz independiente del agente, el cual intercambia acciones y percepciones del entorno y realiza el procesamiento mediante el acceso a dos ficheros, uno donde se encuentra la base de hechos y los elementos específicos del dominio y otro con el resto de la base de conocimiento necesario para el procesamiento de todo el sistema.

Revista Cubana de Ciencias Informáticas
Vol. 11, No. 2, Abril-Junio, 2017
ISSN: 2227-1899 | RNPS: 2301
http://rcci.uci.cu
Pág. 29-45

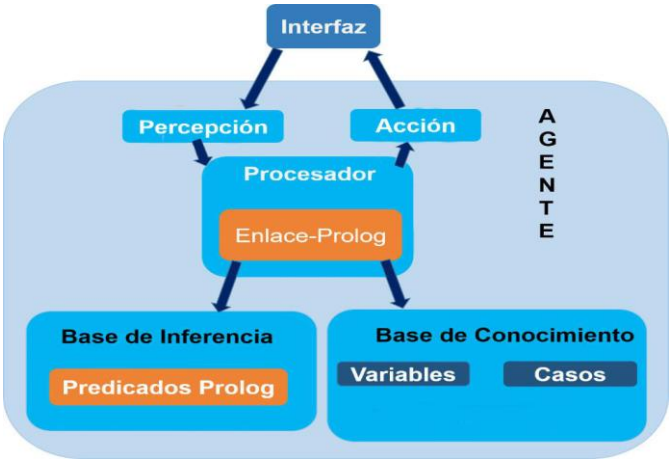

Figura 1: Modelo arquitectónico

Implementación

La programación lógica mediante su mecanismo de unificación compara los valores de las variables e instancia si son iguales, en caso de que alguna de las variables esté vacía toma el valor de la otra variable. Este aspecto se debe tomar en cuenta en el desarrollo de cada uno de los predicados. La recuperación de los casos se realiza utilizando un cuantificador universal (∀), generalmente están predefinidos, uno de los más utilizados es el *findall*. Este predicado permite guardar, en una lista, las variables solicitadas cuando se cumple una condición. Para facilitar el trabajo conviene trabajar con listas y no con las estructuras, el propio Prolog brinda uno de los predicados especiales (=..) para el tratamiento de las estructuras, este permite convertir una lista en una estructura y viceversa.

Cálculo de los pesos: El peso de cada rasgo puede ser calculado mediante un predicado que convierte todos los ejemplos en una lista de listas como se muestra a continuación:

calcularPesoRasgos:- convertirBCenLL(LL),calcularPesosR(LL).

convertirBCenLL(LL):- findall(LR,(hecho(E),E=..[_|[LR]]),LL).

Para el predicado *calcularPesosR(LL)* se pueden seguir las ideas presentadas en (Favero, 2006) para el cálculo de la ganancia de información, adicionando los elementos presentados en el epígrafe sobre el cálculo de los pesos. Sin embargo debido a la complejidad de esta implementación y la evidente afectación al rendimiento del sistema se optó por hacer una consulta de todos los casos de la base de conocimiento y efectuar los cálculos en Java, lo cual arrojó resultados muy favorables. De esta forma solo se realiza la llamada al predicado *convertirBCenLL/1* para ser utilizada la salida en Java.

Revista Cubana de Ciencias Informáticas
Vol. 11, No. 2, Abril-Junio, 2017
ISSN: 2227-1899 | RNPS: 2301
http://rcci.uci.cu
Pág. 29-45

Recuperación de los casos: Siguiendo el algoritmo presentado para la recuperación, primeramente, deben calcularse los valores de creencia del nuevo caso a partir de la información que se tiene en la base de conocimiento, para posteriormente realizar la recuperación de los casos más semejantes:

findall(Ej, (hecho(Ej), semejante(Ej, NuevoC, Crit, Par(VSemj, VCreencia))), ListaSemejantes).

El argumento *Crit* del predicado *semejante/4* se refiere al umbral de semejanza proporcionado por el entorno, o sea, por el usuario que reciba los servicios del agente.

Para la implementación del predicado que calcula la semejanza se transforman las estructuras *ejemplo* en listas, a continuación, se analiza el dominio de cada atributo y se selecciona una función de semejanza adecuada para el cálculo.

semejante(Ej, NuevoC, Crit, Par(VSemj, VCreencia)) :- Ej=..[_|LRE], NuevoC=..[_|LRNC],
comparar(LRE, LRNC, Crit, Par(VSemj, VCreencia, 1)).

El valor 1 al final es para indicar al predicado recursivo *comparar* que se está haciendo la primera llamada. Este predicado además accederá a la lista de atributos almacenados con sus pesos.

Adaptación: Esta lista de casos similares es tratada con Java, donde se lleva a cabo la adaptación luego de que Prolog se haya encargado de realizar todo el proceso de recuperación.

Aprendizaje: El aprendizaje se ejecuta en Java, pero se guarda en Prolog la información de la cantidad de valores de cada atributo, lo cual optimiza los cálculos. Estos valores se actualizan cada vez que se realiza una inferencia, lo cual garantiza tener los valores actualizados a la hora de calcular los valores de creencia a priori $Cree_p$. También se actualizan cuando se adiciona un caso a la base de conocimiento, lo cual permite utilizar dicha información en la actualización de los valores de creencia condicional $Cree_c$.

Pruebas y trabajo futuro

Para validar los resultados del agente se realizaron pruebas de rendimiento con bases de conocimiento de hasta 500 casos con 15 atributos. Los tiempos de respuesta del sistema se muestran en la Figura 2. Como se puede apreciar los resultados en la inferencia son en tiempo real, sin embargo la visualización de los datos se ve más afectada lo cual se debe tomar en cuenta a la hora de implementar el modelo de agente propuesto.

Aunque el objetivo de la implementación realizada fue mostrar el funcionamiento del modelo de agente lógico basado en hechos, se realizaron pruebas básicas para determinar la calidad de las respuestas, obteniéndose resultados aceptables. Como trabajo futuro se propone realizar un estudio de la calidad de los resultados obtenidos por el agente. Para ello se pretenden buscar bases de datos reales e implementar varias funciones para realizar comparaciones entre

ellas, sobre todo para medir la efectividad en el cálculo automático de los pesos y el resultado de los valores de creencia que se obtengan de las respuestas finales.

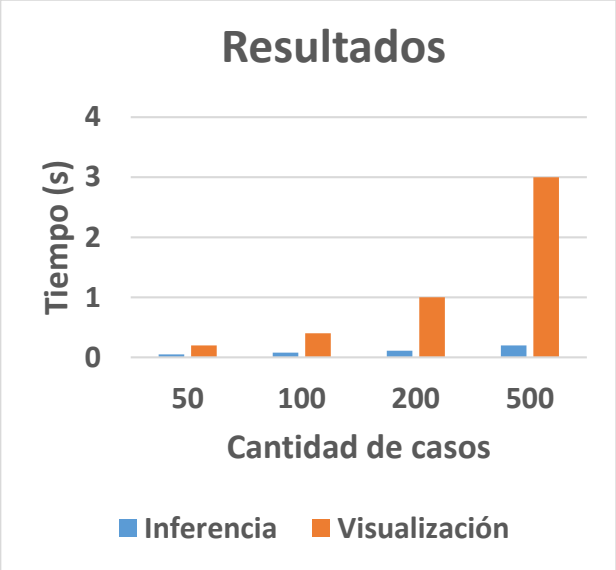

Figura 2. Resultado de las pruebas de estrés

Conclusiones

A partir de la investigación realizada se puede afirmar que el modelo de agente lógico con inferencia basada en hechos es funcional y eficiente, lográndose un prototipo funcional de un agente lógico para la enseñanza de la Inteligencia Artificial en la Universidad de las Ciencias Informáticas. La arquitectura es viable y permite aprovechar las ventajas declarativas del Prolog, con las ventajas visuales y algorítmicas de un lenguaje imperativo, así como las ventajas que incorpora el JPL. El modelo propuesto propone como novedad el cálculo automático de los pesos de los atributos a partir de la teoría de la información y las creencias del agente. Además el tratamiento de la incertidumbre a partir de las creencias del agente tomando en cuenta las probabilidades a priori y condicional.

Referencias

ALI, T. NAJEM, Z. and SAPIYAN, M. A belief revision system for logic programs. Computer Science & Information Technology (CS & IT), 2014: p. 227-231.

ALI, T. NAJEM, Z. and SAPIYAN, M. JPL: Implementation of a Prolog system supporting incremental tabulation. Computer Science and Information Technology (CS & IT), 2016, p. 323-338.

ATANASOVA, I. KRUPKA, J. Architecture and Design of Expert System for Quality of Life Evaluation. Informática Económica, 2013, vol. 17 (3): p. 28-35.

BRATKO, I. Prolog programming for artificial intelligence. Addison-Wesley. Second edition, 1990.

COCA, Y. URS-HTA. Sistema para el diagnóstico de la Hipertensión Arterial usando Razonamiento Basado en Casos en condiciones de incertidumbre. Trabajo de diploma en opción al título de Licenciado en Ciencias de la Computación, UCLV, Santa Clara, 2003.

DE RAEDT, L. Probabilistic programming and its applications. En: 9th International MultiDisciplinary Workshop on Artificial Intelligence. Fuzhou, China, 2015.

DMITRIEV, V.I. Teoría de información aplicada. Moscú, Editorial MIR, 1991. p. 92-100.

DUNSTAN, N. A Hybrid Architecture for Web-based Expert Systems. International Journal of Artificial Intelligence and Expert Systems (IJAE), 2012, vol. 3(4): p. 70-79.

FAVERO, E. L. Programacao em Prolog. Uma abordagem prática. Departamento de Informática. CCEN - UFPA, 2006: p. 243-245.

GONZÁLEZ-ABRIL, L. CUBEOS, F.J. VELASCO, F. ORTEGA, J.A. Ameva: An autonomous discretization algorithm. Expert Systems with Applications, 2009, vol. 36(3), Part 1: p. 5327-5332.

GUOQI, L. YUANXUN, S. SHENG, H. BIN, L. An IPC-based Prolog design pattern for integrating backward chaining inference into applications or embedded systems. Chinese Journal of Aeronautics, 2014, vol. 27(6): p. 1571-1577.

HENDERSON-SELLERS, B. NUMI TRAN, Q.N. DEBENHAM, J. An Etymological and Metamodel-Based Evaluation of the Terms "Goals and Tasksïn Agent-Oriented Methodologies. Journal of Object Technology, 2005, vol. 4(2): p. 131-150.

KADHIM, M. A. AFSHAR ALAM, M. KAUR, H. Design and Implementation of Fuzzy Expert System for Back pain Diagnosis. International Journal of innovate Technology and creative engineering, 2011, vol.1 (9): p. 16-22.

KERDPRASOP, N. and KERDPRASOP, K. Higher Order Programming to Mine Knowledge for a Modern Medical Expert System. IJCSI International Journal of Computer Science, 2011, vol 8 (1): p.64-72.

KURGAN, L. A. and CIOS, K. J. CAIM discretization algorithm. IEEE Transactions on Knowledge and Data Engineering, 2004, vol. 16(2): p. 145-153.

MITCHEL, T. Decision tree learning. En: Machine Learning. McGraw-Hill Science, 1997: p. 52-75.

RUSSELL, S. and NORVING, P. Efficient implementation of logic programs. En: Michael Hirsch. Artificial Intelligence. A Modern Approach. Third Edition. New Jersey: Prentice Hall, 2010a, p. 340-342.

RUSSELL, S. and NORVING, P. The structure of agents. En: Michael Hirsch. Artificial Intelligence. A Modern Approach. Third Edition. New Jersey: Prentice Hall, 2010b: p. 46-58.

RUSSELL, S. and NORVING, P. Applying Bayes' rule: The simple case. En: Michael Hirsch. Artificial Intelligence. A Modern Approach. Third Edition. New Jersey: Prentice Hall, 2010c: p. 496-499.

ZHANG, Q. ZHONG, S. and JIANG, G. Development of an Expert System for Dust Explosion Risk Management Based on ASP.Net and Prolog. Journal of software, 2011, vol. 6 (9): p. 1857-1865.

ZHOU, N. The language features and architecture of B-Prolog. Theory and Practice of Logic Programming, 2012, vol 12(1-2): p. 189-218.

ZUNINO, A. BERDÚN, L. AMANDI, A. JavaLog: un lenguaje para la Programación de Agentes. Revista Iberoamericana de Inteligencia Artificial, 2001, vol 13, p. 94-99.