Modelado del problema de reparto utilizando un enfoque de planificación basado en Answer Set Programming

Adriana Huitzil Tello[1], Claudia Zepeda Cortés[1] y Mauricio Osorio Galindo[2]

[1] Benemérita Universidad Autónoma de Puebla,
Puebla, México
adryhut@gmail.com, czepedac@gmail.com

[2] Universidad de las Américas – Puebla,
Puebal, México
osoriomauri@gmail.com

Resumen La Planificación es una manera en la que un agente puede tomar ventaja del conocimiento que posee, y la habilidad de razonar acerca de las acciones y sus efectos en el ambiente. Answer Set Programming es un lenguaje de programación lógico y declarativo en el área de representación del conocimiento enfocado en la lógica. El propósito de este artículo consiste en usar la Planificación y Answer Set Programming para modelar y resolver el problema de reparto. Consideramos el caso donde un robot se encuentra en una oficina y tiene a su alcance ciertos paquetes, que a través de una secuencia de acciones, debe entregar a los lugares correspondientes de acuerdo a un horario de solicitud.

Palabras clave: Planificación, programación lógica, *Answer Set Programming*.

1. Introducción

La Programación Lógica o LP ofrece una manera más natural de representar conocimiento declarativo utilizando el lenguaje de la lógica matemática [1], construyendo una base de conocimientos a través de hechos y reglas entre diversos objetos que pueden ser capturados en un programa lógico, y con la ayuda de herramientas computacionales poder contestar preguntas sobre los objetos descritos y encontrar soluciones a problemas particulares.

La finalidad de un programa lógico es buscar todos los valores que hacen verdadero a este programa y como resultado de las combinaciones de dichos valores obtener un modelo estable [2]. Para que un programa lógico tenga un significado y pueda ser interpretado por programas computacionales se le debe asignar una semántica, la cual nos permite determinar el tipo de conclusiones que se pueden establecer a partir de un conjunto de reglas [1]. Debido a esto surgen lenguajes que permiten escribir programas lógicos que soportan dichas expresiones tales como el uso de Answer Set Programming. Answer Set Programming (ASP, Programación Lógica Estable o A-Prolog) es un área de representación del conocimiento enfocado en la lógica, basado en lenguajes para el modelado de problemas de cálculo en términos de restricciones

[3, 4]. La técnica de ASP se basa en la posibilidad de representar ciertas colecciones de conjuntos como colecciones de modelos estables [5]. El éxito de la semántica de los modelos estables, y posteriormente de los Answer Sets, se debió en gran medida a su capacidad para resolver problemas. Gracias a la existencia de implementaciones eficientes para calcular modelos estables tales como Smodels[1], DLV[2], Clasp[3], enfocados a resolver problemas en base a la semántica estable, la gama de problemas que podían enfrentarse con este nuevo paradigma creció rápidamente para incluir problemas combinatorios, establecer y resolver problemas de planeación, modelar el comportamiento de agentes lógicos y aplicaciones de inteligencia artificial en general [1].

Un problema de Planificación en el que se puede hacer uso de Answer Set Programming es el problema de Secuenciación Mínima con Tiempos de Realización, que se define de la siguiente manera [6]: Sea *T un conjunto de tareas, para cada tarea t* $\in$ *T, una liberación de tiempo r(t)* $\in$ Z^+, *una longitud l(t)* $\in$ Z^+ *y un peso w(t)* $\in$ Z^+. *Un generador de tareas para T que obedece a un tiempo de realización, es decir, una función f:* $T \rightarrow N$ *dado que, para todo* $u \geq 0$, *si S(u) es el conjunto de tareas t para cada* $f(t) \leq u < f(t) + l(t)$, *entonces* $|S(u)| = 1$ *y para cada tarea t,* $f(t) \geq r(t)$. Como se observa, lo que se establece es un conjunto de tareas, donde a cada tarea se le asigna una liberación de tiempo, además de establecer una longitud y un peso para cada una de ellas.

En este artículo presentamos la solución del problema de Secuenciación Mínima para el caso particular de reparto de oficios basado en un enfoque de Planificación lógica utilizando Answer Set Programming. El problema propuesto de Secuenciación Mínima con Tiempos de Realización que modelamos es el siguiente: *Dado un conjunto de objetos, estos deben ser entregados, a través de una secuencia de acciones, a lugares donde se solicite, en un tiempo indicado, llevando el control de los tiempos en que se realizan las acciones.*

La contribución de este trabajo es modelar el problema, generar una implementación y dar solución al problema de Secuenciación Mínima para un dominio particular (en este caso el reparto de oficios) utilizando un enfoque de planificación basado en Answer Set Programming.

En relación con Answer Set Programming y Planning, actualmente se han realizado diversos proyectos, uno de los más recientes es el modelado de espacio de datos autónomo usando Answer Sets, que está basado en la expresividad de los modelos estables y la acción del lenguaje k para expresar las funciones de gestión de un espacio de datos, el cual se basa en un modelo para especificar un espacio de datos autónomo que son expresados mediante Answer Sets [7]. Por otra parte también se ha aplicado Planeación en Answer Set como es el caso de [8], el cual permite simular y dar apoyo en la generación de nuevos planes de evacuación mediante el uso de Answer Set Programming para describir una zona de riesgo y permitir resolver consultas que no pueden ser resueltas por un Sistema de Información Geográfica (SIG), otro ejemplo de este tipo se muestra en [9], en donde se modelan problemas de evacuación para situaciones de riesgo. Un caso similar al que tratamos en este artículo es

[1] http://www.tcs.hut.fi/Software/smodels/
[2] http://www.dbai.tuwien.ac.at/proj/dlv/
[3] http://www.cs.uni-potsdam.de/clasp/

el Modelado y planeación con Answer Set Programming [10], también se hace uso de preferencias en Answer Set como es el caso de [11] y [12].

En la sección 2 se describen conceptos que se consideran importantes para resolver el problema de reparto de oficios así como algunos trabajos que se relacionan con el trabajo a desarrollar en este artículo. En la sección 3 establecemos una descripción a detalle del problema de reparto de oficios. En la sección 4 se muestra la codificación del problema en el lenguaje A. En la sección 5 se realiza la codificación del problema utilizando la herramienta Smodels. En la sección 6 se indican algunas diferencias de codificación en Clasp con respecto a Smodels, y por último en la sección 7 presentamos las conclusiones.

2. Marco teórico

En esta sección presentamos definiciones básicas para el desarrollo del problema de Secuenciación Mínima.

2.1. Programación lógica

La Programación Lógica trabaja de una forma descriptiva, es decir, estableciendo relaciones entre entidades (objetos del mundo), sin indicar cómo se debe hacer, sino más bien qué hacer, ya que es el programador quien suministra la parte lógica y el intérprete la parte del control [13]. Un programa lógico está formado por un conjunto de cláusulas, que son sentencias descriptivas que puede ser simples o compuestas por otras cláusulas, además de contener operadores binarios, por ejemplo: and (,), or (v), implicaciones (←), negaciones (not, -), etc. [2].

2.2. Lenguaje A

El *lenguaje A* es un elemento muy importante dentro de la Programación Lógica, ya que es el pseudocódigo de un programa para Planning. El alfabeto del lenguaje A consiste de 2 conjuntos disjuntos de símbolos no vacíos que son F y A [3], los cuales corresponden al conjunto de Fluents y al conjunto de Acciones respectivamente.

Los *fluents* forman parte de la descripción del estado del mundo y representan las propiedades de los objetos en el mundo. Un *fluent literal* es un fluent, el cual puede estar precedido por la negación ¬. Las acciones, representan las operaciones aplicadas a los fluents que permiten realizar cambios de estados. El lenguaje de acción A está representado a través de tres sub-lenguajes que son: *lenguaje de descripción del dominio*, *lenguaje de observación* y *lenguaje de consulta* [3, 14].

Lenguaje de descripción del dominio. Indica lo que ocurre entre estados al momento de ejecutar las acciones. La descripción del dominio *D* consiste en proposiciones de efecto que son de la siguiente forma: a **causes** f **if** $p_1,...,p_n$, $\neg q_1,...,\neg q_r$ donde a es una acción, f y $p_1,...,p_n$, $q_1,...,q_r$ son fluents, esto significa que si los fluents literals $p_1,...,p_n$, $\neg q_1,...,\neg q_r$ se cumplen en el estado correspondiente a una situación s, entonces en el estado correspondiente a la situación alcanzada por ejecutar a en s, el fluent literal f debe cumplirse. También se pude incluir ejecutabilidad de condiciones, de la forma: **executable** a **if** $p_1,...,p_n$, $\neg q_1,...,\neg q_r$ donde a es una acción y, $p_1,...,p_n$, $q_1,...,q_r$

son fluents, esto significa que si los fluents literals $p_1,...,p_n$, $\neg q_1,...,\neg q_r$ se cumplen en el estado σ de una situación s, entonces la acción a es ejecutada en s.

Lenguaje de observación: Es un conjunto de observaciones O, formado por valores de proposiciones de la siguiente forma [15]: f **after** $a_1,..,a_m$ en donde f es un fluent y $a_1,..,a_n$ son las acciones. Esta proposición significa que si las acciones $a_1,..,a_m$ se ejecutan en el estado inicial entonces en el estado correspondiente a la situación $[a_1,..,a_m]$ el fluent f se cumple. En otro caso, cuando la secuencia de acciones $a_1,..,a_m$ es una secuencia vacía, únicamente se indica el fluent marcándolo como estado inicial: **initially** f.

Lenguaje de consulta. En un dominio consistente de descripción D, en la presencia de un conjunto de observaciones O implica una consulta Q de la forma: f **after** $a_1,..,a_m$ si para todos los estados iniciales σ_0 que corresponde a (D, O), el fluent literal f se cumple en el estado $[a_m,...,a_1]$ σ_0. Lo cual denotamos como $D \models_O Q$.

2.3. Resolvedores de ASP: Smodels y Clasp

Actualmente, existen varios resolvedores para Answer Set, por cuestiones de espacio solo mencionaremos dos de ellos: Smodels y Potassco[4] en particular la herramienta Clasp, los cuales se utilizarán para obtener los Answer Sets del problema de Secuenciación Mínima. Smodels es una eficiente implementación de la semántica de modelos estables para programas lógicos. Clasp es un sistema solucionador de Answer Set para extender programas lógicos normales.

3. Descripción del problema de reparto de oficios

En esta sección se presenta la descripción para el caso particular del problema de Secuenciación Mínima, el cual consiste en repartir oficios. El problema está enfocado en un robot que tiene como principal tarea llevar oficios a los empleados de las diferentes oficinas. Los elementos que componen el mundo para el problema de reparto de oficios son los siguientes: un robot que es el encargado de entregar los oficios, las oficinas donde se realizarán las entregas y la recepción donde el robot tomará los oficios. Inicialmente, el robot está en la recepción y los oficios se encuentran sobre una mesa los cuales están organizados en carpetas, cada carpeta tiene el nombre del empleado al cual deberán dirigirse los oficios, por lo tanto, si el robot se encuentra en la recepción solo debe tomar los oficios correspondientes, teniendo claro que eso representa la carpeta con el nombre del empleado al que entregará los oficios. Posteriormente el robot deberá entregar los oficios realizando una secuencia de acciones definidas y en base a los horarios que el empleado ha indicado para recibir sus oficios, tomando en cuenta los tiempos que ocupa para trasladarse del estado inicial a cada oficina y viceversa, los tiempos de ida y vuelta deben ser los mismos. El robot solo puede llevar una carpeta de oficios y realizar una entrega en un tiempo determinado.

En este caso se modela la entrega de oficios en una oficina, sin embargo, este modelo puede amoldarse a la entrega de diversos objetos como: herramienta, instrumental, paquetes, sobres, café, etc., y además puede crecer en base al número de oficinas.

[4] http://potassco.sourceforge.net/

Las pruebas que realizamos para la entrega de oficios fueron de una a cinco oficinas, las cuales se muestran en el siguiente sitio: https://sites.google.com/site/codigoentregadeoficios/

Primero se muestra la codificación en Smodels que fue la primera parte con la que trabajamos realizando varias pruebas para obtener una versión final, y posteriormente la codificación en Clasp de estas mismas pruebas, en ambos casos se obtuvieron los Answer Sets de cada prueba para comparar los modelos obtenidos y analizar las variantes de cada una.

4. Código del problema de reparto de oficios en el lenguaje A

En esta sección haremos uso del lenguaje A para modelar el problema de Secuenciación Mínima que hemos planteado anteriormente, el cual consiste en entregar oficios en una oficina:

Fluentes.

`tiempoSalida(T).` %♦Representa el tiempo en que el robot sale de un lugar.
`tiempoLlegada(T).` %Representa el tiempo en que el robot llega a un lugar.
`estaEnLugar(L).` %Indica el lugar en que se encuentra el robot
`robotTieneOficios.` %Indica que el robot tiene los oficios que va a entregar.
`entregoOficiosEnLugar(L).` %El robot ha entregado los oficios en el lugar indicado.

Acciones.

`move(L1,L2).` %Representa que el robot se moverá del lugar L1 al lugar L2.
`tomarOficios.` %Indica que el robot debe tomar los oficios.
`entregarOficios(L,T_entrega).` %Entregar oficios en el tiempo indicado.

Efectos y ejecutabilidad de condiciones.

```
move(L1,L2)
  causes estaEnLugar(L2)
    if holds(estaEnLugar(L1),T), T < length, time(T),
        lugar(L1), lugar(L2), neq(L1,L2).
```

Esta regla significa que el robot se moverá de `L1` a `L2` causando que esté en `L2` si se cumple que el robot se encuentra en `L1` en el tiempo `T`, y `L1` y `L2` son lugares diferentes. Para la ejecución de la acción tomar oficios será de la siguiente manera:

```
tomarOficios
  causes robotTieneOficios
    if holds(estaEnLugar(recepcion),T),
       holds(neg(robotTieneOficios),T), T < length,
```

♦ El símbolo % es usado para indicar un comentario dentro del código

```
      time(T).
```

Esto indica que si el robot ejecuta la acción de tomar oficios causará que el robot tenga los oficios, si se cumple que se encuentra en la recepción y aún no tiene los oficios a entregar. La última ejecución es para la acción entregar oficios:

```
entregarOficios(L,T_entrega)
  causes neg(robotTieneOficios), entregoOficiosEnLugar(L)
    if T < length, time(T), holds(estaEnLugar(L),T),
        holds(robotTieneOficios,T),
        holds(neg(entregoOficiosEnLugar(L)),T),
        holds(tiempo_llegada(T_llegada),T),
        horario_oficios(L,T0,T1), T0<= T_llegada,
        T_llegada <= T1, assign(T_entrega,T_llegada),
        lugar(L), timeh(T_entrega), timeh(T_llegada),
        timeh(T0), timeh(T1).
```

de esta manera se indica que si el robot ejecuta la acción entregar oficios causará que el robot ya no tenga los oficios y que los ha entregado en la oficina, si se cumple que el robot está en el lugar donde realizará la entrega, tiene los oficios a entregar, aún no ha entregado los oficios en ese lugar, se conoce el tiempo en que el robot llegó al lugar para que a través del horario que el empleado indicó se realice una comparación entre el tiempo de llegada y los tiempos establecidos, y si se cumple esta comparación se asigne el tiempo de llegada al tiempo de entrega con el objetivo de no perder el conteo del tiempo.

Estado inicial.

El estado inicial muestra los elementos del mundo antes de sufrir una modificación:

```
initially estaEnLugar(recepcion). %El robot está en la recepción.
initially neg(robotTieneOficios). %El robot aún no tiene los oficios.
initially tiempoSalida(1). %El robot inicia en el tiempo 1.
initially neg(entregoOficiosEnLugar(oA)). %Aún no se han entregado los oficios en el lugar indicado.
```

Estado final.

El estado final representa la meta del plan indicado de esta forma:

```
finally entregoOficioEnLugar(oA). %Entrega de oficios en la oficina.
finally estaEnLugar(oA). %El robot está en la oficina oA.
```

5. Código del problema de reparto de oficios en Smodels

Siguiendo con lo planteado anteriormente, se muestra el código correspondiente a la entrega de oficios en una oficina utilizando Smodels.

Declaración de constantes y tiempos.

```
const length = 4. % Declaración de la constante length.
time(1..length). % Indica el tiempo máximo para llegar a la meta.
const reloj = 4. % Declaración de la constante reloj.
timeh(0..reloj). % Indica la longitud de los pasos para realizar las acciones.
```

Declaración de hechos.

```
lugar(recepcion). % Declaración del lugar recepción.
lugar(oA). % Declaración del lugar oficina de Ana.
horario_oficios(oA,2,3). %Horario de preferencia para recibir los oficios.
tiempo_viaje(recepcion,oA,1). %Tiempo para trasladarse a la oficina.
tiempo_viaje(oA,recepcion,1). %Tiempo para trasladarse a la recepción.
```

Fluentes.

```
fluent(tiempo_salida(T)) :- timeh(T). %Tiempo de salida del lugar.
fluent(tiempo_llegada(T)) :- timeh(T). %Tiempo de llega al lugar.
fluent(esta_en_lugar(L)):- lugar(L). %Lugar en donde está el robot.
fluent(robot_tiene_oficios). %El robot tiene los oficios a entregar.
fluent(entrego_oficios_en_lugar(L)) :- lugar(L). %El robot ha entregado los oficios en el lugar indicado.
```

Acciones.

```
action(move(L1,L2)):- lugar(L1), lugar(L2). %Moverse del lugar L1 al lugar L2.
action(tomar_oficios). %Tomar los oficios.
action(entregar_oficios(L,T_entrega)):- timeh(T_entrega), lugar(L). %Hacer entrega de los oficios en el tiempo de entrega establecido.
```

Estado inicial.

```
initially(esta_en_lugar(recepcion)).%El robot está en la recepción.
initially(neg(robot_tiene_oficios)). %El robot no tiene los oficios.
initially(tiempo_salida(1)). %Indica que el robot inicia en el tiempo 1.
initially(neg(entrego_oficios_en_lugar(oA))). %Aún no se han entregado los oficios en el lugar indicado.
```

Estado final.

El estado final representa la meta del plan indicado de esta manera:

```
finally(entrego_oficios_en_lugar(oA)). %Entrega de oficios en la oficina oA.
finally(esta_en_lugar(oA)). %El robot está en la oficina oA.
```

Declaración de las condiciones de ejecutabilidad.

```
executable(move(L1,L2),T) :- T < length, time(T),
          holds(esta_en_lugar(L1),T),
          lugar(L1), lugar(L2), neq(L1,L2).
```

Esta regla significa que el robot se moverá de `L1` a `L2` si se cumple que el robot se encuentra en `L1` en el tiempo `T` y `L1` y `L2` son lugares diferentes. Para la ejecución de la acción tomar oficios será de la siguiente manera:

```
executable(tomar_oficioS,T) :- T < length, time(T),
          holds(esta_en_lugar(recepcion),T),
          holds(neg(robot_tiene_oficios),T).
```

esto indica que el robot ejecuta la acción de tomar oficios si se cumple que el robot se encuentra en la recepción y aún no tiene los oficios. La última ejecución es la acción de entregar oficios:

```
executable(entregar_oficios(L,T_entrega),T):- T < length,
        time(T), lugar(L), timeh(T_entrega),
        holds(esta_en_lugar(L),T),
        holds(robot_tiene_oficios,T),
        holds(neg(entrego_oficios_en_lugar(L)),T),
        holds(tiempo_llegada(T_llegada),T),
        horario_oficios(L,T0,T1), T0<= T_llegada,
        T_llegada <= T1, assign(T_entrega,T_llegada),
        timeh(T_llegada), timeh(T0), timeh(T1).
```

de esta manera, indicamos que el robot ejecuta la acción entregar oficios si se cumple que el robot está en el lugar donde realizará la entrega, tiene los oficios a entregar, y aún no ha entregado los oficios en ese lugar, se conoce el tiempo en que el robot llegó al lugar para que a través del horario que el empleado indicó se realice una comparación entre el tiempo de llegada y los tiempos establecidos, y si se cumple esta comparación se asigne el tiempo de llegada al tiempo de entrega con el objetivo de no perder el conteo del tiempo.

Reglas de causa.

```
causes_dd(esta_en_lugar(L2),move(L1,L2),
esta_en_lugar(L1)):-   lugar(L1), lugar(L2).
```

Esta regla significa que si el robot está en el lugar `L1` y se ejecuta la acción mover de `L1` a `L2` entonces causa que el robot esté en el lugar `L2`. El caso contrario de esta regla es:

```
causes_dd(neg(esta_en_lugar(L1)),move(L1,L2),
esta_en_lugar(L1)) :- lugar(L1), lugar(L2), neq(L1,L2).
```

que quiere decir, que si el robot está en el lugar `L1` y se ejecuta la acción mover de `L1` a `L2` entonces causa que el robot ya no esté en ese el lugar `L1`. Existen otras re-

glas de causa de la misma forma que la anterior para la acción mover, en donde el efecto es obtener el tiempo de llegada y negar de igual forma el tiempo de llegada y salida. Esto se podrá observar en el sitio ya mencionado debido a que no se tiene más espacio para escribirlas.

Para la acción tomar oficios tenemos la siguiente regla de causa:

```
causes(tomar_oficios, robot_tiene_oficios).
```

esta regla indica que si el robot toma los oficios causará que el robot tenga los oficios a entregar. Para la acción de entregar oficios tenemos dos reglas de la misma forma que se hace para la acción tomar oficios, en donde negamos que el robot tiene los oficios y que ya entregó los oficios, esto podemos observarlo en el sitio mencionado.

6. Código del problema de reparto de oficios en Clasp

El código correspondiente de la aplicación en Clasp es muy parecido al de Smodels, de tal forma que por cuestiones de espacio, solo se mostrarán las diferencias que tiene Clasp con respecto a Smodels, las cuales son:

Declaración de constantes y tiempos.

```
#const length = 4. % Declaración de la constante length.
time(1..length).  % Indica el tiempo máximo para llegar a la meta.

#const reloj = 4.  % Declaración de la constante reloj.
timeh(0..reloj). % Indica la longitud de los pasos para realizar las acciones.
```

Declaración de hechos.

```
lugar(recepción;oA). %Declaración de 2 lugares recepción y oficina de Ana.
```

La declaración de fluentes, acciones, estados iniciales y finales y reglas de causa se hace de la misma manera que en Smodels.

Declaración de las condiciones de ejecutabilidad.

En este caso, se utilizarán reglas llamadas restricciones de cardinalidad, que indican las posibles combinaciones que se pueden hacer en un conjunto de valores. Éstas se definen de la siguiente forma:

```
1{executable(tomar_oficios,T) : T < length: time(T) }1.
executable(tomar_oficios,T) :-
                        holds(esta_en_lugar(recepcion),T),
                        holds(neg(robot_tiene_oficios),T).
```

esta regla de ejecución indica tomar oficios en un tiempo `T`, donde `T` sea menor a la constante `length` y `T` sea de tipo `time`, si el robot está en la recepción y no tiene los oficios. Para la ejecución de la acción entregar oficios se define de manera similar,

indicando entre llaves los tipos de las variables a utilizar para la ejecución. Para ejecutar la acción mover se tiene lo siguiente:

```
1{executable(move(L1,L2),T) : T < length: time(T) : lugar(L1): lugar(L2)} 1.
```

lo que indica, que el robot se moverá de `L1` a `L2 en el tiempo T, donde L1` y `L2` son lugares diferentes.

Para el caso de 5 oficinas se podrá observar con mayor precisión el uso de las condiciones de ejecutabilidad. Las soluciones de ambas implementaciones se observan en el sitio descrito anteriormente donde se encuentran todos los códigos en ambas versiones junto con sus ejecuciones, las cuales muestran de una manera más clara los resultados obtenidos en cada implementación.

7. Conclusiones

Al ejecutar estos códigos en Smodels y Clasp, se observó que la entrega de oficios de una hasta cinco oficinas se realiza satisfactoriamente y para el tiempo de resolución de Answer Set, Clasp los obtiene más rápido. Por otra parte, la forma de codificar en Smodels y Clasp es muy parecida, pero Clasp permite realizar una reducción de código. En Smodels fue un tanto difícil codificar porque no se conocía a fondo la sintaxis de esta herramienta y en Clasp fue más fácil por la experiencia adquirida.`

Agradecimiento. Este trabajo ha sido apoyado por el Proyecto de Ciencia Básica del Fondo Sectorial SEP-CONACyT con número de registro 101581.

Referencias

1. Juan Antonio Navarro: Lógica Aplicada a Answer Sets. Tesis de Licenciatura. Universidad de las Américas Puebla. Online http://www.mpi-sws.org/~jnavarro/papers/uthesis.pdf (última fecha de verificación mayo 2012), páginas 61 (2003)
2. Guillermo Didier Bravo: Desarrollo y modelado de algoritmos para la genotipificación de secuencias: el caso de los transplantes, alelos HLA. Tesis de Licenciatura. Universidad de las Américas Puebla. Online http://catarina.udlap.mx/u_dl_a/tales/documentos/lis/bravo_a_gd/capitulo3.pdf (última fecha de verificación mayo 2012), páginas 77 (2005)
3. Chitta Baral: Knowledge Representation, Reasoning and Declarative Problem Solving with Answer Sets. Cambridge University Press, Cambridge (2003)
4. Iikka Niemelä: Logic programming with stable model semantics as a constraint programming paradigm. Annals of Mathematics and Artificial Intelligence 25(3-4) 241–273 (1999)
5. http://www.ia.urjc.es/cms/sites/default/files/userfiles/file/ia3/teoria/Intro-ASP.pdf
6. G. Ausiello, P. Crescenzi, G. Gambosi, V. Kann, A. Marchetti Spaccamela and M. Protasi: Complexity and Approximation. Combinatorial Optimization Problems and Their Approximability Properties. Springer, páginas 436 (2003)
7. Gabriela Montiel Moreno, Zechinelli Martini José Luis and Genoveva Vargas Solar: Modelling autonomic dataspaces using answer sets. Inteligencia Artificial. Revista Iberoamericana de Inteligencia Artificial, Volumen 14. Número 48. Online http://redalyc.uaemex.mx/src/inicio/ArtPdfRed.jsp?iCve=92513175002 (2010)

8. Claudia Zepeda and David Sol: Evacuation Planning using Answer Set Programming: An initial approach. http://www.engineeringletters.com/issues_v15/issue_2/EL_15_2_10.pdf (última fecha de verificación mayo 2012) (2012)
9. José Miguel Leyva Cruz: Representación de Planes de Evacuación usando Answer Set Programming. Tesis de Ingeniería. Universidad Tecnológica de la Mixteca. Online http://jupiter.utm.mx/~tesis_dig/10107.pdf (última fecha de verificación mayo 2012), páginas 155 (2007)
10. Matilde Hernández Salas: Modelado y planeación con ASP (Answer Set Programming). Tesis de Maestría. Universidad de las Américas Puebla, 2004. Online http://catarina.udlap.mx/u_dl_a/tales/documentos/msp/hernandez_s_m/portada.html (última fecha de verificación mayo 2012) (2004)
11. Diego Rodríguez Villanueva: Utilización de los enfoques para Preferencias en Answer Set Programming para definir familias de problemas con preferencias. Tesis de Ingeniería. Universidad Tecnológica de la Mixteca. Online http://jupiter.utm.mx/~tesis_dig/10300.pdf (última fecha de verificación mayo 2012) (2007)
12. Alexander Gelbukh and Sergio Suárez Guerra: Special Issue: Advances in Computer Science and Engineering, 23:3-13. Online http://www.micai.org/rcs/Vol23.pdf (2006)
13. Ravi Sethi: Programming Languages: Concepts & Constructs. 2nd Ed. Addison-Wesley (1997)
14. Michael Gelfond and Vladimir Lifschitz: Action languages. Electron. Trans. Artif. Intell. 2:193-210 (1998)
15. V. Subrahmanian and C. Zaniolo: Relating stable models and AI planning domains. In: Proceedings of the 12th International Conference on Logic Programming, L. Sterling. Pages 233-247. Tokyo, Japan, MIT Press (1995)