Una Extensión de Graphplan con Aprendizaje y Argumentación para la Elección de Acciones

Diego García
dgarcia@cs.uns.edu.ar

Alejandro García
agarcia@cs.uns.edu.ar

Laboratorio de Investigación y Desarrollo en Inteligencia Artificial (LIDIA)[1]
Departamento de Ciencias e Ingeniería de la Computación
Universidad Nacional del Sur.
Av. Alem 1253, (8000) Bahía Blanca, Argentina

Resumen

En este trabajo se presenta una extensión al algoritmo de Graphplan que durante la resolución de un problema de planificación genera conocimiento automáticamente (experiencia) que podrá ser utilizado para resolver nuevos problemas del mismo dominio de planificación. El conocimiento generado puede ser utilizado para mejorar el algoritmo de Graphplan tanto en la fase de expansión del grafo, como en la elección de acciones durante la fase de búsqueda del plan.

Palabras clave: Inteligencia Artificial. Planificación. Graphplan. Aprendizaje.

1 Introducción

Planificar eficientemente requiere de heurísticas de búsqueda específicas a cada dominio de planificación. Sin embargo, construir heurísticas apropiadas para cada nuevo dominio es una tarea difícil. Graphplan [1] se ha convertido en uno de los planificadores de propósito general más promisorios. Sin embargo, la búsqueda que realiza sobre el grafo de planificación podría mejorarse. La alternativa que se presenta en este trabajo es extender el algoritmo de Graphplan para que durante la resolución de un problema de planificación genere conocimiento automáticamente (experiencia), el cual podrá ser utilizado para resolver nuevos problemas del mismo dominio de planificación.

La extensión propuesta incorpora a Graphplan la capacidad de generar conocimiento automáticamente durante la fase de búsqueda. El conocimiento generado tendrá la forma de *reglas rebatibles* a favor o en contra de seleccionar un conjunto de acciones, dependiendo del éxito o fracaso que hayan tenido estos conjuntos al ser seleccionados durante la búsqueda. Estas reglas pueden ser utilizadas luego por el planificador para guiar la búsqueda de un plan para un nuevo problema. El conjunto de reglas rebatibles generadas define un *programa lógico rebatible* y puede utilizarse un mecanismo de argumentación rebatible (por ejemplo DeLP [2]), para razonar y generar argumentos a favor y en contra de elegir acciones en determinadas situaciones de la búsqueda.

Este trabajo está organizado de la siguiente manera. En la sección siguiente se presenta una breve descripción del algoritmo de Graphplan. Luego, en la sección 3 se describirá la idea de la extensión propuesta. En la sección 4 se mostrará la forma en que el planificador genera

[1] Miembro del Instituto de Investigación en Ciencia y Tecnología Informática (IICyTI)

y utiliza el conocimiento. En particular se ejemplificarán los siguientes aspectos: generación de reglas a favor y en contra de seleccionar acciones, utilización del conocimiento para guiar la fase de búsqueda y para controlar la expansión del grafo.

2 Graphplan

El algoritmo de Graphplan [1] está basado en el paradigma de "análisis del grafo de planificación". Utiliza una estructura llamada *grafo de planificación* que codifica el problema de planificación de manera que muchas restricciones inherentes al problema se manifiestan explícitamente, reduciendo así la búsqueda en el grafo.

Un grafo de planificación (ver Figura 1) es un grafo dirigido con dos tipos de nodos organizados en niveles: los niveles pares contienen nodos de proposición (representados con un círculo), esto es, precondiciones o efectos de acciones. El nivel 0 consiste precisamente de las proposiciones que son verdaderas en el estado inicial del problema de planificación. Los niveles impares contienen nodos de acción (representados con rectángulos) que corresponden a instancias de acciones cuyas precondiciones están presentes en el nivel anterior, y cuyos efectos están en el nivel siguiente. El grafo tiene tres tipos de arcos que representan relaciones entre las acciones y las proposiciones. Los nodos de acción en un nivel de acción i están conectados a sus precondiciones en el nivel de proposición $i-1$ por *arcos de precondición*, y a sus efectos en el nivel $i+1$ por *arcos de agregación* (add-edges) representados por líneas sólidas y *arcos de eliminación* (delete-edges) representados por líneas punteadas. Existe un tipo especial de acción llamado *no-op* o *frame action* (cuadrado sólido), que representa la posibilidad que una proposición se mantenga inalterada de un nivel de proposición i al siguiente $i+2$.

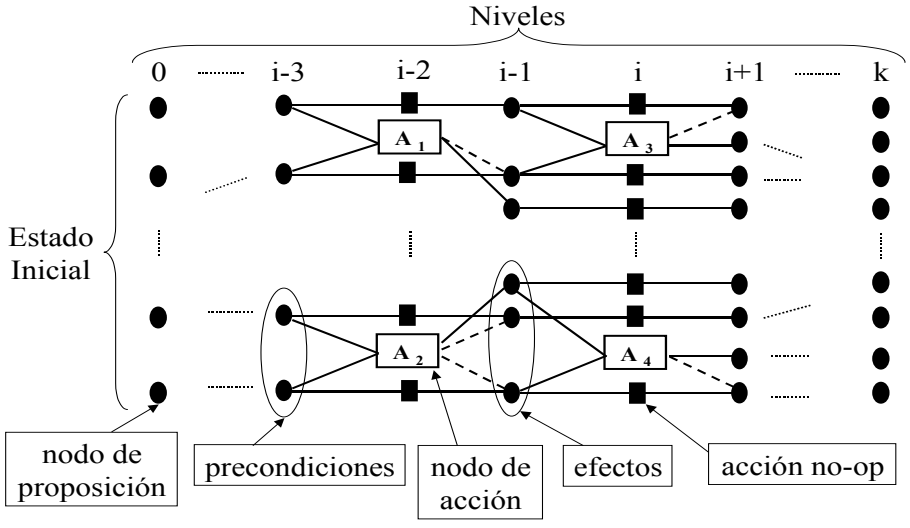

Figura 1: Esquema del grafo de planificación

Como puede observarse en la Figura 1, el grafo representa acciones "paralelas" en cada nivel de acción. Sin embargo, que dos acciones estén presentes en el mismo nivel de acción, no significa que puedan ser ejecutadas simultáneamente, pues pueden ser *mutuamente excluyentes* o *mutex*. La relación de *mutex* se define recursivamente como sigue (ver Figura 2).

Dos acciones son *mutex* en el nivel i, si:

- **Efectos inconsistentes:** los efectos de una acción, eliminan parte de lo que agregan los efectos de la otra acción.
- **Interferencia:** los efectos de una acción eliminan las precondiciones de la otra.
- **Precondiciones excluyentes:** las acciones tienen precondiciones que son mutuamente excluyentes (Soporte inconsistente) en el nivel anterior $i-1$.

Dos proposiciones son *mutex* en el nivel i, si:

- **Soporte inconsistente:** todas las acciones cuyos efectos agregan estas proposiciones, son mutex entre si en el nivel anterior $i-1$.

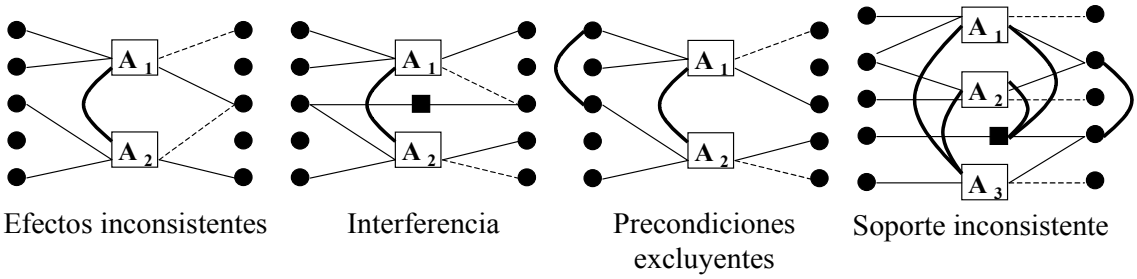

Figura 2: Relaciones de Exclusión Mutua (líneas curvas)

El algoritmo alterna entre dos fases: expansión del grafo y búsqueda del plan. Durante la fase de expansión, el grafo de planificación se extiende hacia adelante en el "tiempo", hasta lograr una condición necesaria (aunque insuficiente) para la existencia de un plan. En la fase de búsqueda del plan se realiza una búsqueda hacia atrás (backward-chaining) sobre el grafo generado hasta el momento, buscando un plan que resuelva el problema. Si se encuentra un plan, el proceso termina, de lo contrario, el ciclo se repite expandiendo el grafo más niveles.

En la fase de expansión el grafo se extiende a partir del último nivel de proposición i, agregando: el siguiente nivel de acción $i+1$, correspondiente a todas las acciones cuyas precondiciones están en el nivel i, y el nivel de proposición $i+2$, correspondiente a los efectos de la acciones del nivel $i+1$. La condición necesaria para la existencia de un plan, se logra cuando todas las proposiciones del estado final (metas) están presentes en el último nivel del grafo y no son mutex unas con otras.

La búsqueda comienza en el último nivel de acción del grafo y procede hacia el nivel de proposición 0. En cada nivel de acción i del grafo se debe seleccionar un subconjunto S de las acciones (incluidas *no_op*) presentes en el nivel i que cumplan las metas del nivel $i+1$. Además las acciones en S no deben ser mutex entre sí el nivel i. Luego se debe seleccionar un subconjunto S' de acciones del nivel $i-2$ que cumplan las metas del nivel $i-1$,representadas por las precondiciones de las acciones en S. Si el conjunto S' existe, se continúa hacia atrás hasta llegar al nivel 0. De lo contrario, si S' no existe en $i-2$ se debe seleccionar un subconjunto alternativo en i (backtracking).

Note que Graphplan genera planes parcialmente ordenados, es decir, las acciones elegidas en cada nivel de acción pueden ejecutarse en cualquier orden y lograrán el mismo efecto. Por

ejemplo, supongamos que el plan se encontró en un grafo expandido hasta el nivel 4. En el nivel de acción 1 se eligió la acción A_1, y en el nivel de acción 3 las acciones A_2 y A_3. Luego el plan se representará como una lista de conjuntos de la siguiente manera: $[\{A_1\}, \{A_2, A_3\}]$, esto es, A_1 debe ejecutarse antes que A_2 y A_3, pero A_2 y A_3 pueden ejecutarse en cualquier orden o simultáneamente. También es importante notar que aunque las acciones *no_op* son consideradas en la elección de acciones durante la búsqueda, no son consideradas parte del plan.

3 Extensión propuesta

La propuesta de este trabajo es incorporar a Graphplan la capacidad de generar conocimiento automáticamente durante la fase de búsqueda, con el fin de utilizarlo para mejorar la performance del planificador en problemas futuros. El conocimiento generado tendrá la forma de *reglas rebatibles* a favor o en contra de seleccionar un conjunto de acciones. Estas reglas pueden ser utilizadas luego por el planificador para guiar la búsqueda de un plan para un nuevo problema. El conjunto de reglas rebatibles generadas define un *programa lógico rebatible* y puede utilizarse un mecanismo de argumentación rebatible (por ejemplo DeLP [2, 3]), para razonar y generar argumentos a favor y en contra de elegir acciones en determinadas situaciones de la búsqueda.

Durante la búsqueda de un plan, un punto importante a tener en cuenta es cuando se produce backtracking. El backtracking en un nivel de acción i indica que el subconjunto de acciones A elegido en el nivel i no fue el correcto para lograr las metas del nivel $i+1$. Sin embargo, es posible que sólo un subconjunto de acciones $S \subseteq A$ sea responsable del backtracking. Este subconjunto puede ser identificado a través de las relaciones de exclusión mutua que se producen durante la búsqueda, en niveles de acción anteriores a i. Una vez identificado este subconjunto, la propuesta es generar una regla rebatible que indique que subconjunto de acciones *no* conviene elegir para lograr las metas del nivel $i+1$. En este caso, la regla rebatible tendrá la forma "$\sim select(S) -\!\prec m_1, .., m_k$ ", donde S es el subconjunto identificado y $m_1, .., m_k$ son las metas del nivel $i+1$.

En el caso que un conjunto de acciones A elegido en un nivel i resulte satisfactorio (forme parte del plan) para lograr las metas del nivel $i+1$, se generará una regla que indica que es conveniente elegir A para lograr las metas del nivel $i+1$. En este caso, la regla rebatible tendrá la forma "$select(A) -\!\prec m_1, .., m_k$ ", donde $m_1, .., m_k$ son las metas del nivel $i+1$.

Supongamos que el planificador ha generado en problemas anteriores un conjunto Δ de reglas rebatibles, y que Π son las metas del nivel en el cual se encuentra la búsqueda en un nuevo problema. Entonces $\mathcal{P}$=$\Pi \cup \Delta$ define un programa lógico rebatible formado por un conjunto de hechos Π y el conjunto de reglas rebatibles Δ. A partir de $\mathcal{P}$, es posible construir un *argumento* $\mathcal{A}$ a favor de un literal "$select(A)$", si es posible derivar $select(A)$ con algún subconjunto de hechos y reglas rebatibles de $\mathcal{P}$. Así como es posible construir un argumento $\mathcal{A}$ a favor de un literal, también es posible construir *contra-argumentos* que si resultan *mejores* que $\mathcal{A}$ se convierten en *derrotadores* para $\mathcal{A}$. En DeLP un literal está *garantizado* cuando el argumento que lo sustenta no tiene derrotadores, o todos sus derrotadores están a su vez derrotados. Mayores detalles sobre DeLP pueden encontrarse en [2, 3].

En esta aplicación en particular, decir que un literal $select(A)$ está garantizado, significa que de acuerdo al conocimiento adquirido por el planificador, resulta conveniente seleccionar

el conjunto de acciones A para lograr las metas. De esta forma, cuando el planificador se enfrente a un nuevo problema de planificación, puede utilizar las reglas generadas en búsquedas anteriores como una heurística para guiar la elección de acciones en cada nivel. Como se explicará más adelante, el conocimiento generado también puede utilizarse para decidir entre iniciar la búsqueda o expandir el grafo más niveles durante la fase de expansión.

4 Generación automática de conocimiento y planificación guiada por la experiencia

En esta sección se mostrará a través de una serie de ejemplos, la forma en que el planificador genera y utiliza conocimiento. En particular se ejemplificarán los siguientes aspectos:

- Generación de reglas *a favor* de seleccionar acciones.
- Generación de reglas *en contra* de seleccionar acciones.
- Utilización del conocimiento para guiar la fase de búsqueda.
- Utilización del conocimiento para controlar la expansión del grafo.

El dominio de planificación elegido es una variación del problema conocido en la literatura como *dinner-date problem* [7], y consiste en preparar una cena sorpresa para alguien que se encuentra durmiendo en la casa. Las acciones posibles de este dominio de planificación son: *cook*, *wrap*, *washH*, *carry*, *dolly*. En la Figura 3 puede verse una especificación de las acciones en notación STRIPS.

Action	Preconditions	add-effects	delete-effects
cook	*cleanH*	*dinner*	
wrap	*quiet*	*present*	
washH	*dirtyH*	*cleanH*	*dirtyH*
carry	*garbIn*	*garbOut* *dirtyH*	*cleanH*
dolly	*garbIn*	*garbOut*	*quiet*

Figura 3: Especificación STRIPS del problema de la Cena Sorpresa

La acción *cook* requiere como precondición las manos limpias (*cleanH*) para poder ser ejecutada y produce la cena (*dinner*). *Wrap* es la acción de envolver un regalo (*present*) y tiene como precondición silencio en la casa (*quiet*), ya que el regalo es una sorpresa y no queremos despertar al receptor. *WashH* es la acción de lavarse las manos, requiere que las manos estén sucias (*dirtyH*), y logra que las manos estén limpias (*cleanH*). Por último las acciones *carry* y *dolly* son dos formas alternativas de sacar la basura (*garbOut*) y ambas requieren que la basura este dentro de la casa (*garbIn*). Como consecuencia de realizar la acción *carry* las manos quedarán sucias por el contacto directo con la basura. Con la acción *dolly* se utiliza un carro para sacar la basura, lo cual evita ensuciarse las manos pero produce mucho ruido y se rompe el silencio de la casa.

A continuación se presenta una secuencia de ejemplos, donde se va generando conocimiento automáticamente, y este conocimiento es utilizado en los ejemplos siguientes. En el ejemplo 1 se presenta un problema sencillo donde el planificador genera una regla rebatible a favor de seleccionar un conjunto de acciones. Luego, en el ejemplo 2 se muestra como a raíz del backtracking en la fase de búsqueda se genera una regla rebatible en contra de seleccionar un conjunto de acciones. Además, una vez que el plan es encontrado, se generan dos reglas más. En el ejemplo 3 las cuatro reglas generadas en los ejemplos 1 y 2 son utilizadas para guiar la búsqueda en un nuevo problema y el plan es encontrado directamente, sin necesidad de backtracking. Por último, en ejemplo 4 se muestra como puede utilizarse el conocimiento generado para controlar la expansión del grafo, más precisamente, para decidir entre expandir más niveles o iniciar la búsqueda.

Ejemplo 1 : Consideremos dentro de este dominio un problema sencillo que, comenzando con las manos limpias y la casa en silencio, tiene como meta tener la cena lista y el regalo envuelto. El estado inicial esta representado por $\{cleanH, quiet\}$ y el estado final por $\{dinner, present\}$. Asumamos que el planificador no cuenta con conocimiento de resolver problemas anteriores, en cuyo caso se comportará de la misma forma que el algoritmo original de graphplan. En la Figura 4 puede verse el grafo expandido para este problema. En este caso, el grafo es expandido hasta el nivel 2 ya que en este nivel están presentes las metas $\{dinner, present\}$ y no son mutex entre sí. Luego se inicia la fase búsqueda (parte resaltada en la Figura 1) que encuentra el siguiente plan: [$\{cook, wrap\}$].

Como mencionamos anteriormente, si la elección de un conjunto de acciones resulta satisfactorio durante la búsqueda, se genera una regla rebatible a favor de elegir dicho conjunto. En este caso, se genera la siguiente regla:

$$(R_1)\ select(\{cook, wrap\}) -\!\prec dinner, present$$

donde $\{dinner, present\}$ son las metas del nivel (2 en este caso) donde fue elegido el conjunto de acciones $\{cook, wrap\}$. Esta regla representa el siguiente conocimiento: *"Si las metas son* $\{dinner, present\}$ *existen razones para elegir el conjunto de acciones* $\{cook, wrap\}$ *".*

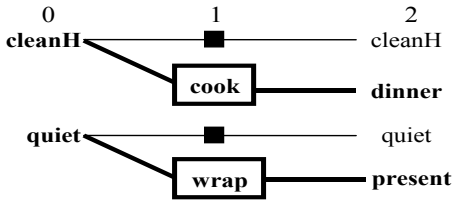

Figura 4: Grafo de planificación del Ejemplo 1.

Ejemplo 2 : Consideremos ahora un problema un poco mas complejo. Inicialmente, al igual que el Ejemplo 1 la casa esta en silencio y las manos están limpias, pero en este caso hay basura dentro de la casa. Las metas son: tener la cena lista, el regalo envuelto y además la basura fuera de la casa. El estado inicial está representado entonces por $\{cleanH, quiet, garbIn\}$ y el estado final por $\{dinner, present, garbOut\}$. En la Figura 5 puede verse el grafo expandido hasta el nivel 2. Como en este nivel están presentes las metas $\{dinner, present, garbOut\}$ y no son mutex entre sí, entonces se inicia la fase búsqueda.

Utilizando el programa lógico rebatible formado por la regla R_1 (el conocimiento generado en el ejemplo 1), y tomando como hechos las metas $\{dinner, present, garbOut\}$, existe un

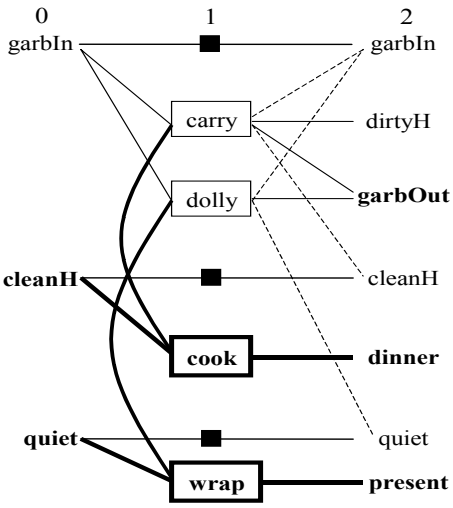

Figura 5: Expansión del grafo hasta el nivel 2 y búsqueda (parte resaltada)

argumento para $select(\{cook, wrap\})$. Por lo tanto, el planificador elige primero las acciones *cook* y *wrap* que logran las metas *dinner* y *present*. Sin embargo, la búsqueda del plan fracasará porque no existe un subconjunto de acciones que no sea mutex y que logre todas las metas. Note que para lograr *dinner* la única alternativa es *cook* y para lograr *present* la única es *wrap*. Para lograr *garbOut* hay dos alternativas: *carry* o *dolly*, pero son mutex con *cook* y *wrap* respectivamente. Como no es posible encontrar un plan, el grafo es expandido dos niveles más, como se muestra en la Figura 6.

La fase de búsqueda se inicia ahora en el nivel 4, y al igual que antes, el conocimiento representado por la regla R_1 proporciona un argumento para elegir primero las acciones *cook* y *wrap*. A diferencia de la búsqueda anterior, ahora contamos con otra alternativa para lograr la meta restante *garbOut*, que es elegir la acción *no_op*. Por lo tanto, el conjunto de acciones elegidas en el nivel 3 es $\{cook, wrap\}$. La búsqueda continúa a partir del nivel 2, con las metas { garbOut, cleanH, quiet }. La única forma de lograr las metas *cleanH* y *quiet* es utilizando *no_op*, ya que son verdaderas en el estado inicial. Para lograr *garbOut*, otra vez hay dos posibilidades (*dolly* y *carry*) y las dos son mutex con las acciones *no_op* elegidas (ver Figura 6). Por lo tanto no hay forma de lograr las metas { garbOut, cleanH, quiet } en el nivel 2, y debe seleccionarse un conjunto alternativo de acciones en el nivel 3 (backtracking).Como mencionamos anteriormente, el backtracking indica que el conjunto $\{cook, wrap\}$ elegido en el nivel 3 no es el adecuado, y podemos generar automáticamente una regla rebatible para capturar esta situación.

Para generar la regla, primero debemos identificar cuales de las acciones dentro del conjunto $\{cook, wrap\}$ son las que provocaron el backtracking. Esto se logra a través de las relaciones de exclusion mutua del nivel de acción 1. En la Figura 6 podemos ver que la acción *carry* en el nivel 1, es mutex con la acción *no_op* que logra la meta *cleanH*. Esto nos indica que la acción *cook* del nivel 3, que tiene como precondición *cleanH*, es una de las responsables del backtracking. Lo mismo ocurre para la acción *wrap* del nivel 3, ya que *dolly* en el nivel 1 es mutex con la acción *no_op* que logra *quiet* (una de las precondiciones de *wrap*). Por lo tanto el conjunto de acciones identificado como responsables del backtracking es { cook, wrap }. Luego se genera la siguiente regla:

$$(R_2) \sim select(\{cook, wrap\}) \prec dinner, present, garbOut$$

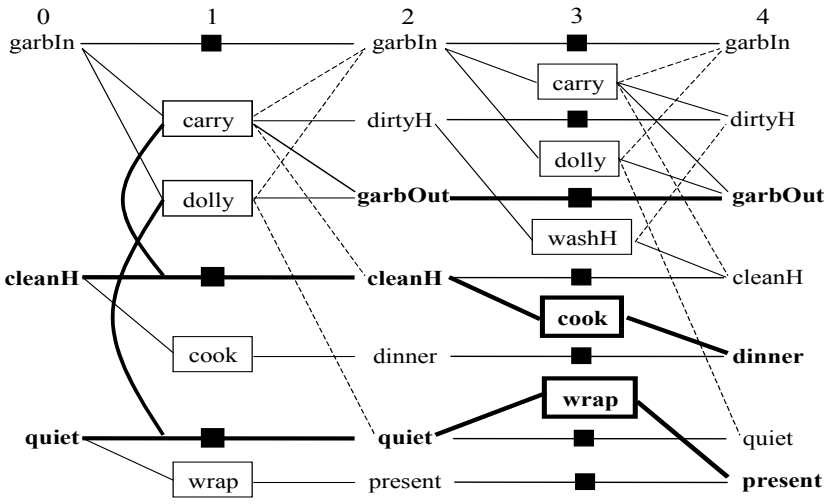

Figura 6: Expansión del grafo hasta el nivel 4 y búsqueda (parte resaltada)

Esta regla representa el siguiente conocimiento: *"Si las metas son* $\{dinner, present, garbOut\}$ *existen razones para* **no** *elegir el conjunto de acciones* $\{cook, wrap\}$ *".*

Nótese que la regla R_2 (en contra de seleccionar $\{cook, wrap\}$) contradice a la regla R_1 (*a favor* de seleccionar $\{cook, wrap\}$). Como se explicó antes, la regla R_1 expresa que *"Si las metas son* $\{dinner, present\}$ *existen razones (rebatibles) para elegir* $\{cook, wrap\}$ *".* La regla R_2 refina este conocimiento, indicando que ante la presencia de las metas *dinner*, *present* y *garbOut*, entonces existen razones para **no** elegir "$\{cook, wrap\}$". Esto es, el cuerpo de R_1 describe una situación diferente al descripto por el cuerpo de R_2, el cual tiene una meta más: *garbOut*. De esta manera, si se cumple $\{dinner, present\}$, entonces se tendrá un argumento para $select\{cook, wrap\}$. En cambio, si se cumple $\{dinner, present, garbOut\}$, entonces habrá un argumento (más específico) para $\sim select\{cook, wrap\}$ que derrota al argumento a favor de $select\{cook, wrap\}$ (ver Ejemplo 3). Más detalles sobre la derrota entre argumentos pueden encontrase en [2].

Después de generar la regla se continúa con la búsqueda, seleccionando un conjunto de acciones alternativo en el nivel 3. Como el conocimiento generado no sugiere la selección de otras acciones, se puede continuar la búsqueda utilizando cualquier heurística de selección. En la figura 7 se muestra uno de los planes posibles $[\{wrap\}, \{dolly, cook\}]$ encontrado al continuar la búsqueda.

En el nivel 3 se eligió el conjunto de acciones $\{dolly, cook\}$ para lograr las metas $\{dinner, present, garbOut\}$ del nivel 4, y en el nivel 1 se eligió el conjunto de acciones $\{wrap\}$ para lograr las metas $\{garbIn, cleanH, present\}$ del nivel 2. Por lo tanto se generan las siguientes reglas rebatibles, que se agregan al conocimiento del planificador:

$$(R_3)\ select(\{dolly, cook\}) -\!\prec dinner, present, garbOut$$
$$(R_4)\ select(wrap) -\!\prec garbIn, cleanH, present$$

Ejemplo 3 : En este ejemplo mostraremos como el conocimiento generado en los ejemplos 1 y 2 guía al planificador en la búsqueda de un plan para un nuevo problema. El estado inicial de este nuevo problema esta representado por $\{garbIn, dirty, quiet\}$, es decir, inicialmente la casa

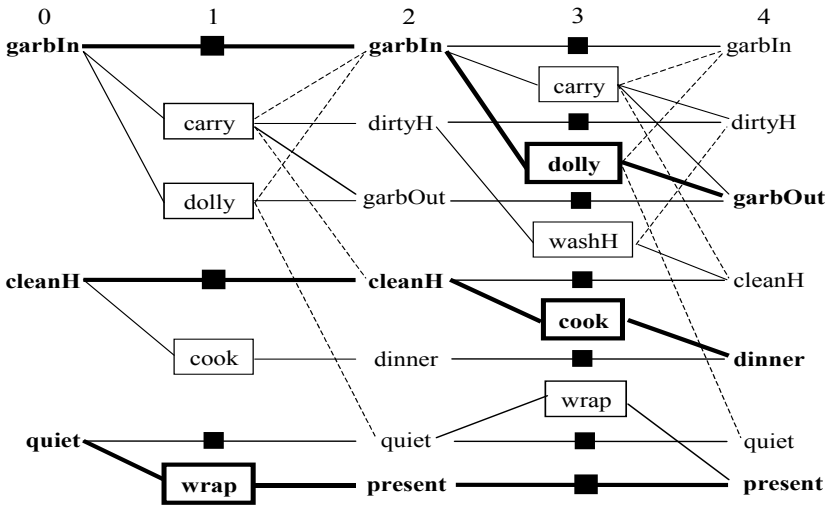

Figura 7: Uno de los planes posibles (parte resaltada) para el ejemplo 2

esta en silencio, las manos sucias y la basura dentro de la casa. Las metas del problema son $\{dinner, present, garbOut\}$, es decir, tener la cena lista, el regalo envuelto y la basura fuera de la casa. Como puede verse en la Figura 8, el grafo se expande hasta el nivel 4, por ser el primer nivel donde todas las metas están presentes y no son mutex entre si, y luego se inicia la fase de búsqueda.

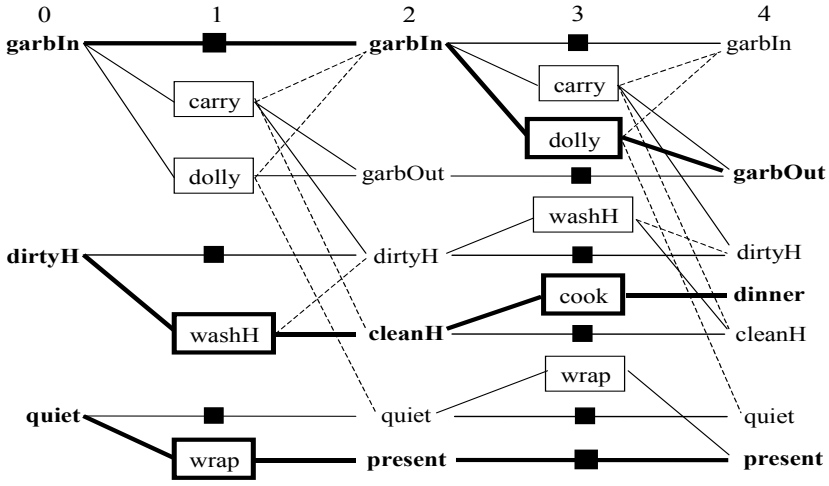

Figura 8: Grafo de planificación del Ejemplo 3 y búsqueda (parte resaltada)

Consideremos el programa lógico rebatible formado por las reglas (R_1),(R_2),(R_3) y (R_4) (conocimiento generado en los ejemplos 1 y 2), y tomando como hechos las metas $\{dinner, present, garbOut\}$. El único literal garantizado es $select(\{dolly, cook\})$ sustentado por el argumento no derrotado $\{select(\{dolly, cook\}) -\!\prec dinner, present, garbOut\}$, generado con la regla $(R3)$. También existe un argumento para $select(\{cook, wrap\})$ generado con la regla $(R1)$, pero es derrotado por un argumento en contra generado por la regla $(R2)$. Por lo tanto el planificador elegirá primero las acciones *dolly* y *cook*, que satisfacen las metas *dinner* y *garbOut*. Para lograr la meta restante *present*, la única opción es seleccionar la acción *no_op*, ya

que $wrap$ es mutex (por interferencia) con la acción ya seleccionada $dolly$. Luego el planificador seleccionará el conjunto $\{dolly, cook\}$ para lograr las metas del nivel 4.

La búsqueda continúa en el nivel 2, siendo ahora las metas las precondiciones de las acciones elegidas, esto es $\{garbIn, cleanH, present\}$. Para poder utilizar el conocimiento en la elección de las acciones debemos actualizar los hechos del programa, debido a que ahora las metas son otras. Consideremos entonces el programa formado por las reglas (R_1),(R_2),(R_3) y (R_4) y tomando ahora como hechos las metas $\{garbIn, cleanH, present\}$. Como existe un argumento no derrotado para $select(wrap)$ generado con las regla $(R4)$, el planificador elegirá primero la acción $wrap$ que logra la meta $present$. Para lograr las metas restantes $garbIn$ y $cleanH$, las únicas alternativas son elegir la acción no_op para $garbIn$ y la acción $washH$ para $cleanH$. Por lo tanto el planificador seleccionará el conjunto $\{washH, wrap\}$ para lograr las metas del nivel 2. Las precondiciones $\{garbIn, dirtyH, quiet\}$ de las acciones elegidas constituyen las metas del nivel 0. Como estas proposiciones están presentes en el estado inicial, las acciones elegidas constituyen un plan: $[\{washH, wrap\}, \{dolly, cook\}]$. Es importante destacar que el plan fue encontrado directamente, sin necesidad de backtracking, debido a que la búsqueda fue guiada por el conocimiento adquirido en los problemas anteriores.

Ejemplo 4 : Por medio del siguiente ejemplo mostraremos como puede utilizarse el conocimiento generado para controlar la expansión del grafo, mas precisamente, para decidir entre expandir más niveles o iniciar la búsqueda. El estado inicial del problema elegido es $\{garbIn, cleanH, quiet\}$, es decir, la basura esta en la casa, las manos limpias y la casa esta en silencio. La meta es $\{garbOut, cleanH, dinner, present\}$, esto es, la basura fuera de la casa ,las manos limpias, la cena lista y el regalo envuelto.

El grafo se expande inicialmente hasta el nivel 2. Como las metas están presentes y no son mutex entre sí, el algoritmo original de graphplan iniciaría la fase de búsqueda. Sin embargo, esta búsqueda fallará porque en este ejemplo no es posible obtener un plan de un grafo de dos niveles. Esta búsqueda infructuosa resulta una pérdida de tiempo y sólo sirve para detectar que el grafo debe ser expandido más niveles. El conocimiento generado puede utilizarse para detectar esta situación y así expandir el grafo lo suficiente antes de realizar una búsqueda infructuosa.

Tomando el programa lógico rebatible formado por las reglas (R_1),(R_2),(R_3) y (R_4) y tomando como hechos las metas $\{garbOut, cleanH, dinner, present\}$ existe un argumento no derrotado para $\sim select(\{cook, wrap\})$ generado con la regla (R2). Note además que (ver Figura 9) la única forma de lograr las metas $dinner$ y $present$ en el nivel 2 es a través de las acciones $cook$ y $wrap$. Por lo tanto, contamos con un argumento para no seleccionar las acciones $cook$ y $wrap$, que son las únicas alternativas para lograr las metas $dinner$ y $present$ en el nivel 2. Esto es una clara indicación de que el grafo debe ser expandido más niveles, para que haya más alternativas. Luego, gracias al conocimiento adquirido, la fase de búsqueda en el nivel 2 no se realiza y el grafo se expande directamente hasta el nivel 4. A diferencia del nivel 2, en el nivel 4 existen mas alternativas para lograr $diner$ y $present$ (las acciones no_op) y el plan es encontrado.

5 Conclusiones

La extensión propuesta, incorpora al algoritmo de Graphplan la capacidad de generar conocimiento automáticamente durante la resolución de un problema y utilizarlo para resolver

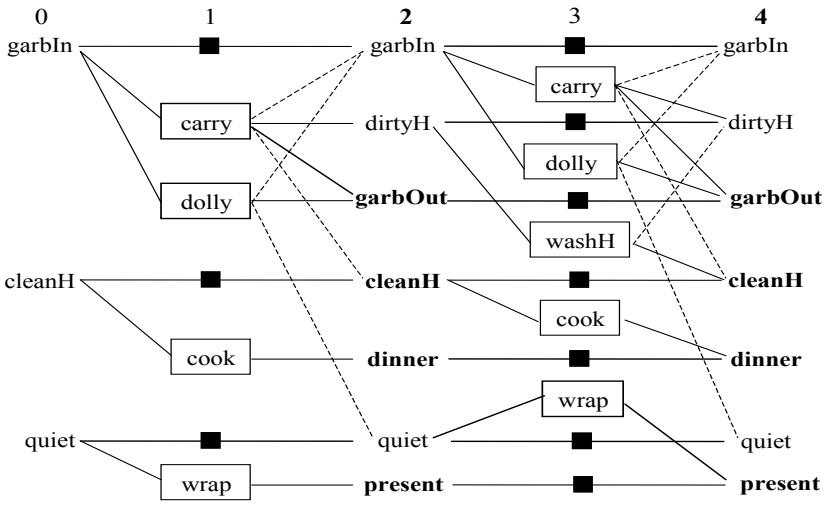

Figura 9: Grafo de planificación del Ejemplo 4

problemas futuros. Como se mostró a través de los ejemplos de la sección 4, la utilización del conocimiento adquirido mejora el algoritmo de Graphplan tanto en la fase de expansión como en la fase de búsqueda.

Como el conocimiento es generado automáticamente por el planificador al resolver problemas, su rendimiento mejora a medida que nuevos problemas son resueltos. Esto permite que el planificador sea "entrenado" resolviendo problemas sencillos y adquiera conocimiento que mejore su rendimiento en problemas más complejos.

Para probar el funcionamiento de la extensión propuesta en este trabajo se realizó una implementación en Prolog con resultados satisfactorios.

Referencias

[1] A. L. Blum and M. L. Furst. *Fast Planning through planning graph analysis* Journal of Artificial Intelligence, 90 (1-2):281-300, 1997.

[2] Alejandro J. García. *Programación en Lógica Rebatible, Definición, Semántica Operacional y Paralelismo.* (`http:/cs.uns.edu.ar/~ajg`) Tesis Doctoral. Universidad Nacional del Sur. 2000.

[3] Alejandro J. García and Guillermo R. Simari. Defeasible logic programming: An argumentative approach. *Theory and Practice of Logic Programming*, To appear. http://xxx.lanl.gov/format/cs.AI/0302029.

[4] Diego García y Alejandro García *Planificación con Aprendizaje de Reglas Rebatibles para elección de Acciones utilizando Argumentación* IV Workshop de Investigadores en Ciencias de la Computación. WICC'02. Universidad Nacional del Sur, Bahía Blanca, 2002.

[5] Diego García y Alejandro García *Extendiendo Graphplan con Técnicas de Aprendizaje* V Workshop de Investigadores en Ciencias de la Computación. WICC'03. Universidad Nacional del Centro de la Provincia de Buenos Aires, Tandil, 2003.

[6] R. Kambhampati, E. Lambrecht and E. Parker. *Understanding and Extending Graphplan* In Proc. 4th European Conference on Planning. September 1997.

[7] Daniel S. Weld. Recent advances in AI planning. *AI Magazine*, 20(2):93–123, 1999.