Rulando

Alfonso Gastalver Llamas

Alumno de Inteligencia Artificial 1, Ingeniería Informática

Universidad de Sevilla

alfonsogastalverllamas@gmail.com

**Abstract.** Este trabajo trata sobre grafos de planificación. Trataremos de encontrar un plan o secuencia de acciones para llegar hasta un objetivo final partiendo desde un estado inicial.

Usaremos el formalismo PDDL y, a partir de él, representaremos el problema y lo solucionaremos usando un algoritmo de grafos de planificación: el algoritmo GRAPHPLAN.

1. Introducción

Rulando es un programa diseñado en el lenguaje de programación Lisp, y trata desde un marco dentro del campo de inteligencia artificial, la planificación para resolver problemas descritos en formalismos PDDL.

Para ello usaremos una implementación de un algoritmo ya diseñado anteriormente por dos científicos de la computación: Avrim Blum y Merrick Furst. En 1995, estas dos personas desarrollaron el algoritmo “GRAPHPLAN”, como un algoritmo escrito en STRIPS que planeaba una secuencia de acciones desde un punto inicial para llegar a un objetivo.

El nombre del programa ha sido elegido como “Rulando” debido a que se trata de ir abordando acciones a través y re-eligiendo aquellas que nos lleven al objetivo sin que entre ellas haya problemas de exclusión. Así que se asemeja a una especie de ir probando y probando hasta dar con un plan válido.

Para ayudar al lector de esta documentación explicaré como va a realizarse. La siguiente sección abordará unos conocimientos que necesitaremos para poder entender cómo funciona todo esto y qué queremos conseguir, en concreto: Formalismos PDDL, Algoritmos de planificación y búsqueda backtracking. La siguiente sección tratará sobre la explicación del algoritmo realizado y comentaremos un poco su funcionamiento en LISP. Más tarde abordaremos los Experimentos obtenidos, para ello hablaremos de tres problemas de planificación originales y los resultados que nos devuelve el programa. Y el último apartado será una conclusión sobre todo lo realizado.

1. Preliminares

A continuación se explicarán unos cuantos conceptos que nos serán muy útiles a la hora de entender el programa y todo lo referente a problemas de planificación.

* 1. Formalismo PDDL:

El formalismo PDDL o por su nombre completo: “Planning Domain Description Language”, es un estándar de representación de dominios de problemas de planificación dentro del marco de la Inteligencia Artificial.

Este estándar fue desarrollado por Drew McDermott en 1998 inspirandose en STRIPS y en ADL. Desde entonces, ha evolucionado mucho y se ha usado en muchos aspectos. Pero nosotros veremos la versión básica, que representa un problema de planificación de la siguiente manera:

* Constantes (Mayúsculas): Objetos del Mundo.
* Variables (Minúsculas): Cualquier objeto del problema.
* Símbolos de predicado: Propiedades de los objetos.
* Símbolos de acciones: Operadores.

Con esto podemos definir el siguiente nivel del formalismo:

* Átomos: fórmulas de la forma P(o1,…,o2). Siendo P un predicado y sus argumentos constantes o variables.
* Literales: átomos o negación de átomos.
* Cerrados: Si no contienen variables.

Y así ya podemos empezar a definir el problema teniendo en cuenta la “Hipótesis del mundo cerrado”: Cualquier átomo que no se menciones se supone falso. Por tanto, podemos definir estados, operadores y objetivos:

* Estados: Lista de literales cerrados. Se define “estado inicial” como el estado que tenemos antes de aplicar ningún operador.
* Objetivos: Estado que puede variables en sus literales, y se satisface cuando nos encontramos en un estado que cumple sus átomos positivos y ninguno de sus negativos.
* Operadores: Son acciones que se puede ejecutar en base a unas precondiciones y que generan unos efectos. Las precondiciones son un conjunto de literales que deben estar en el estado actual y los efectos son literales que modifican el estado actual cuando se aplica el operador.

Por tanto se define un problema como un estado-inicial, un objetivo y unos operadores. Y un plan (solución) se define como la secuencia a seguir de acciones para que partiendo desde el estado inicial lleguemos a cumplir el objetivo. Pueden haber planes que no lleguen al objetivo pero no serán solución a nuestro problema.

* 1. Algoritmos de planificación

Un algoritmo de planificación es un conjunto de instrucciones o reglas bien definidas, ordenadas y finitas que permite resolver un problema de planificación mediante pasos sucesivos que no generen dudas a quien deba realizar dicha actividad. Dados un estado inicial y una entrada, siguiendo los pasos sucesivos se llega a un estado final y se obtiene una solución.

En la historia se han usado dichos algoritmos para muchas cosas y principalmente en el campo de la computación para la planificación de procesos del ordenador y en nuestro campo para la inteligencia artificial.

Esto ha tenido mucha relevancia ya que nos permite llegar a planificar fácilmente muchos planes para llegar a objetivos que a priori puede resultar difícil de ver o de entender. Y así tenemos que los algoritmos de planificación han sido muy utilizados en nuestra época.

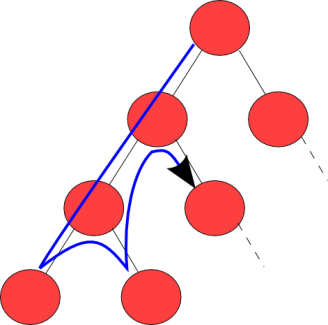
* 1. Búsqueda backtracking

Esta búsqueda la usaremos para extraer nuestra solución del grafo de planificación encontrado y por tanto es preciso explicarla antes para poder comprenderla.  Backtracking es una estrategia para encontrar soluciones a problemas que satisfacen restricciones.

En su forma básica, la idea de backtracking se asemeja a un recorrido en profundidad dentro de un grafo dirigido. El grafo en cuestión suele ser un árbol, o por lo menos no contiene ciclos. Sea cual sea su estructura, existe sólo implícitamente. El objetivo del recorrido es encontrar soluciones para algún problema. Esto se consigue construyendo soluciones parciales a medida que progresa el recorrido; estas soluciones parciales limitan las regiones en las que se puede encontrar una solución completa. El recorrido tiene éxito si, procediendo de esta forma, se puede definir por completo una solución. En este caso el algoritmo puede bien detenerse (si lo único que se necesita es una solución del problema) o bien seguir buscando soluciones alternativas (si deseamos examinarlas todas). Por otra parte, el recorrido no tiene éxito si en alguna etapa la solución parcial construida hasta el momento no se puede completar. En tal caso, el recorrido vuelve atrás exactamente igual que en un recorrido en profundidad, eliminando sobre la marcha los elementos que se hubieran añadido en cada fase. Cuando vuelve a un nodo que tiene uno o más vecinos sin explorar, prosigue el recorrido de una solución.

Esencialmente, la idea es encontrar la mejor combinación posible en un momento determinado, por eso, se dice que este tipo de algoritmo es una [búsqueda en profundidad](http://es.wikipedia.org/wiki/B%C3%BAsqueda_en_profundidad). Durante la búsqueda, si se encuentra una alternativa incorrecta, la búsqueda retrocede hasta el paso anterior y toma la siguiente alternativa. Cuando se han terminado las posibilidades, se vuelve a la elección anterior y se toma la siguiente opción (hijo [si nos referimos a un árbol]). Si no hay más alternativas la búsqueda falla. De esta manera, se crea un árbol implícito, en el que cada nodo es un estado de la solución (solución parcial en el caso de nodos interiores o solución total en el caso de los nodos hoja).

Normalmente, se suele implementar este tipo de algoritmos como un procedimiento [recursivo](http://es.wikipedia.org/wiki/Recursividad). Así, en cada llamada al procedimiento se toma una variable y se le asignan todos los valores posibles, llamando a su vez al procedimiento para cada uno de los nuevos estados. La diferencia con la [búsqueda en profundidad](http://es.wikipedia.org/wiki/B%C3%BAsqueda_en_profundidad) es que se suelen diseñar funciones de cota, de forma que no se generen algunos estados si no van a conducir a ninguna solución, o a una solución peor de la que ya se tiene. De esta forma se ahorra espacio en memoria y tiempo de ejecución.

 Imagen de algoritmia Backtracking.

1. Algoritmos

Rulando utiliza el algoritmo GraphPlan para resolver problemas de planificación así que dicho algoritmo es el que explicaremos aquí.

El algoritmo graphplan consiste de dos partes:

* La expansión de la gráfica.
* La extracción de la solución.

La fase de expansión extiende un grafo de planificación(planning graph) hacia adelante hasta encontrar una condición necesaria (pero insuficiente) para la existencia del plan. La fase de extracción de la solución ejecuta una búsqueda hacia atrás (backtracking) en la gráfica en busca de un plan que resuelva el problema; si no se encuentra ninguna solución, entonces se repite el ciclo de expansión de la gráfica.

El grafo de planificación contiene dos tipos de nodo, nodos de literales y nodos de acción colocados en niveles. Los niveles pares contienen nodos de literales, el nivel cero contiene las condiciones iniciales del problema y los niveles impares corresponden a las instancias de las acciones. Hay un solo nodo por cada acción cuyas precondiciones están presentes (y son mutuamente consistentes) en el nivel anterior. Los segmentos conectan los nodos de literales con las acciones (en el siguiente nivel).

Durante la fase de expansión el algoritmo va creando capas de acciones y de literales añadiendo los enlaces de exclusión mutua entre ellos, para llegar a la capa de acciones deben cumplirse las precondiciones de las acciones y además deben estar también las acciones de persistencia (dejar la cosa como está). Y para llegar a la siguiente capa de literales deben tenerse en cuenta los efectos de las acciones realizadas así como las de persistencia.

En cada capa de literales comprobará si hemos cumplido el objetivo sin exclusiones mutuas entre ellas y si es así extraerá la solución realizando un backtracking hacia atrás para organizar un plan de solución que no tenga problemas de exclusión mutua entre sus acciones, precondiciones y efectos. Es preciso realizar backtracking pues si el algoritmo va hacia adelante y se equivoca debe poder tomar otro camino.

Por tanto falta definir qué son los enlaces de exclusión mutua de los que tanto hemos hablado. Pues bien, se conocen como enlaces mutex, e implican que la ejecución de una acción no es compatible con otra con la que esté enlazada a la vez. Y lo mismo para los literales, no podemos encontrarnos en un estado real en el que tengamos dos literales con enlaces mutex.

Tipos en enlaces mutex:

* Acciones:
  + Interferencia: Un efecto de una niega una precondición de otra.
  + Efectos Inconsistentes: Un efecto de una es la negación de otra.
  + Necesidades que compiten: Tienen precondiciones que se excluyen mutuamente.
* Literales:
  + Soporte inconsistente: uno es la negación de otro.
  + Efectos excluyentes: cualquier par de acciones que los producen son mutuamente excluyentes.

Definidos los enlaces podemos empezar a definir los pasos del algoritmo:

Recibe como argumento: estado-inicial, objetivo y operadores.

1. Crear capa literales 0 a partir del estado inicial
2. Hacer i=0
3. Repetir:
   1. Si en Capa i están los literales del objetivo sin mutex:
      1. Hacer RES el resultado de extraer un plan del grafo mediante backtracking.
      2. Si RES no es fallo devolver RES y terminar.
   2. Hacer i=i+1
   3. Crear capa acciones i cuyas precondiciones están en capa literales i-1 sin enlaces mutex. Crear mutex de acciones i.
   4. Crear capa literales i con los efectos de las acciones de la capa acciones i. Crear mutex de literales i.
   5. Conectar los enlaces de acciones i con literales de i-1 y literales de i según precondiciones y efectos. (Para extraer plan).

Hasta que acciones i sea igual a literales i y acciones i-1 igual a literales i-1 (incluyendo mutex)

1. Devolver Fallo.

En Lisp tenemos creado este algoritmo de la siguiente forma:

;;;;;;;;;;;;;;;;;;;; ALGORITMO GRAPHPLAN ;;;;;;;;;;;;;;;;;;;;

(defun graphplan (estado-inicial objetivo operadores)

(let\* ((acciones ())

(literales estado-inicial)

(enlaces-la ())

(enlaces-al ())

(mutex-a ())

(mutex-l ())

(grafo-planificacion (list (list acciones literales enlaces-la enlaces-al mutex-a mutex-l)))

(capa-actual 0)

(a-previa ())

(l-previa ())

(la-previa ())

(al-previa ())

(ma-previa (list () () ()))

(ml-previa (list () ()))

(res 0))

(write-line "Comenzando algoritmo...")

(loop while (and (equal res 0) (or (not (equal (nth 0 (nth capa-actual grafo-planificacion)) a-previa)) (not (equal (nth 1 (nth capa-actual grafo-planificacion)) l-previa))))

do (let\* ()

(write-line (concatenate 'string "Ejecutando capa " (write-to-string capa-actual)))

(if (comprobar-objetivo (nth 1 (nth capa-actual grafo-planificacion)) objetivo ml-previa)

(let\* ()

(write-line "Objetivo encontrado. Ejecutando búsqueda de plan...")

(setf res (extrae-plan grafo-planificacion objetivo))))

(if (equal res 0)

(let\* ()

(setf a-previa (nth 0 (nth capa-actual grafo-planificacion)))

(setf l-previa (nth 1 (nth capa-actual grafo-planificacion)))

(setf la-previa (nth 2 (nth capa-actual grafo-planificacion)))

(setf al-previa (nth 3 (nth capa-actual grafo-planificacion)))

(setf ma-previa (nth 4 (nth capa-actual grafo-planificacion)))

(setf ml-previa (nth 5 (nth capa-actual grafo-planificacion)))

(setf capa-actual (+ capa-actual 1))

(let\* ((a-actual (crea-a operadores l-previa ml-previa))

(l-actual (crea-l a-actual))

(la-actual (crea-la a-actual))

(al-actual (crea-al a-actual))

(ma-actual (crea-ma a-actual ml-previa))

(ml-actual (crea-ml ma-actual l-actual)))

(setf grafo-planificacion (append grafo-planificacion (list (list a-actual l-actual la-actual al-actual ma-actual ml-actual)))))))))

(write-line "Fin de algoritmo.")

(loop for a in res collect (nth 2 a))))

Por otro lado la extracción de la solución se basará en realizar una búsqueda backtracking hacia atrás mediante recursión y eligiendo en cada caso las acciones que cumplan los requisitos mutex entre ellas, sus precondiciones y sus efectos.

Como resultado tendremos un plan de secuencias de acciones o más, posibles que nos dirán como solucionar nuestro problema de planificación.

1. Experimentación

Hemos elegido 3 problemas de planificación en PDDL con los que probar este algoritmo.

* 1. Descripción de los experimentos

**Primer Problema: El Problema de Mike.**

Mike es una persona que sin quererlo ni tomarlo trabajaba como policía en una comisaría, salió de casa a las 8.00 a.m. de la mañana para ir a trabajar y para su horror se encontró que el mundo estaba lleno de zombies, es decir criaturas no muertas que buscan carne humana y cerebros. Mike, no quiere morir pero tiene un problema: Su pistola está descargada. Mike tiene balas en el bolsillo y sabe cargar el arma y disparar. Se ha encontrado con un zombie con pinta de vagabundo que va a hacia él. Mike, que es fan de las películas de zombies sabe que le tiene que disparar en la cabeza y lo reconoce como tal al instante. El zombie se está acercando pero aún está lejos con lo que le da tiempo a pensar un plan para realizar.

Su objetivo: Muerto (Zombie)

Sus Acciones:

* Matar(Zombie): Precondiciones (Cargada(Pistola)) Efectos (-Pistola(Cargada), Muerto(Zombie))
* Cargar(Pistola) : Precondiciones (Tener(Balas)) Efectos(-Tener(Balas))

Su estado inicial: Tener(Balas)

Es un problema de planificación sencillo.

Si ejecutamos este problema en nuestro algoritmo tendremos la siguiente secuencia tras eliminar las acciones de persistencia: Cargar(Pistola) -> Matar(Zombie).



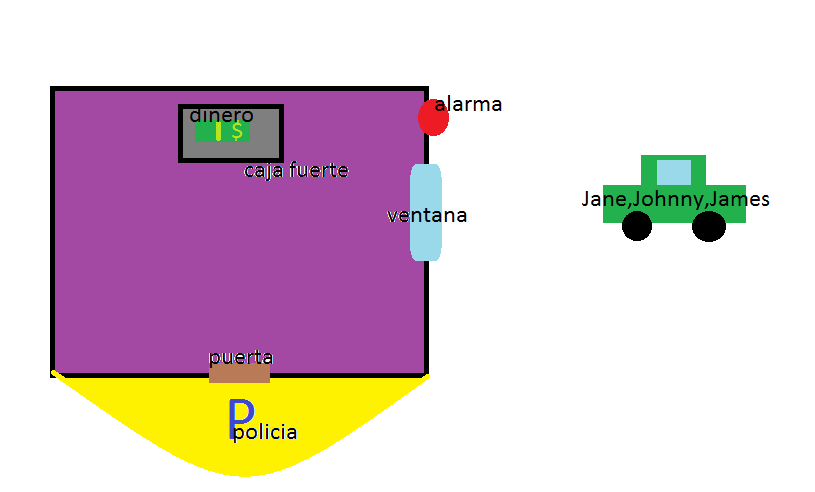
**Segundo Problema: El Problema del Salchichón.**

A Manolo le encanta el salchichón así que decide crearse su propio salchichón para ello sabe que tiene que tener primero un cerdo pero luego no sabe muy bien qué hacer si tiene que alimentarlo o matarlo o convertirlo en salchichón se ha hecho un lío entero y decide usar a “Rulando” para que le planifique un plan de cómo conseguir su tan ansiado salchichón.



**Tercer Problema: El Problema de los ladrones.**

En este problema un grupo de ladrones intentan robar un banco. John, James y Jane siempre van juntos porque tienen miedo a que les pillen así que irán juntos a todos lados. Han deducido que la mejor forma de robar el banco es no acercarse cerca del guarda así que evitarán siempre esa acción y si no pueden realizarla preferirán no robar el banco. Podemos ver el esquema en la siguiente imagen.



Este problema queda definido en Lisp de la siguiente forma:

(defvar \*estado-inicial\* (list (list "+" "estan-en-coche") (list "-" "estan-en-alarma") (list "-" "estan-en-ventana") (list "-" "estan-en-caja-fuerte") (list "-" "alarma-desactivada") (list "-" "ventana-abierta") (list "-" "caja-fuerte-abierta") (list "-" "dinero-robado")))

(defvar \*objetivo\* (list (list "+" "estan-en-coche") (list "+" "dinero-robado")))

(defvar \*operadores\* (list \*ir-coche\* \*ir-ventana\* \*ir-alarma\* \*ir-caja-fuerte\* \*desactivar-alarma\* \*abrir-ventana\* \*abrir-caja-fuerte\* \*robar-dinero\* \*estan-en-coche\* \*no-estan-en-coche\* \*estan-en-alarma\* \*no-estan-en-alarma\* \*estan-en-ventana\* \*no-estan-en-ventana\* \*estan-en-caja-fuerte\* \*no-estan-en-caja-fuerte\* \*alarma-desactivada\* \*alarma-no-desactivada\* \*ventana-abierta\* \*ventana-no-abierta\* \*caja-fuerte-abierta\* \*caja-fuerte-no-abierta\* \*dinero-robado\* \*dinero-no-robado\*))

* 1. Resultados obtenidos

En cuanto al primer problema “Rulando” funciona a la perfección, en este caso nos devuelve la siguiente secuencia de acciones: “Cargar-Pistola” 🡪 “Matar-Zombie”

Con lo que Mike, conseguirá salvarse del zombie y podrá sobrevivir, por suerte estaba en su poder nuestro programa “Rulando” y ha podido salvarse.

En cuanto al segundo problema, Manolo también conseguirá su salchichón. Con lo cual “Rulando” le fue de maravilla.

Sin embargo para el tercer problema genera el grafo de planificación completo pero a la hora de extraer solución no encuentra una, esto se debe a que con las persistencias la solución que encuentra posee mutex y vuelve a atrás para coger otra y luego al volver atrás por lo mismo vuelve a coger la de antes haciendo por tanto un bucle. No he podido solucionar aún este problema.

1. Conclusiones y trabajo futuro

En conclusión podemos decir que el trabajo de resolver problemas de planificación usando el algoritmo graphplan es completo pues siempre encuentra una solución si existe y sino, devuelve fallo. No obstante el programa “Rulando” aún necesita alguna modificación con lo que en un futuro podríamos añadir funciones extras y arreglar algunos fallos que nos da.

Este proyecto podría resultar muy útil en muchas aplicaciones no sólo en el de la informática que es el que nos acontece ahora mismo. Por tanto podemos decir como conclusión final que estos problemas se pueden aplicar a muchas competencias como pueden ser problemas para edificación o otros muchísimos campos.

Por mi parte he aprendido bastante sobre grafos de planificación, formalismos PDDL y el algoritmo graphplan, y me puede resultar útil a la hora de elaborar planes de acciones más complicados que los que hemos resuelto aquí y que no se puedan resolver fácilmente. Hay muchos más algoritmos apartes de graphplan para resolver estos problemas, POP es un algoritmo que también sirve para planificar y por tanto podemos comprobar que hay muchas soluciones a este tipo de problemas distintas y por muchos caminos.

Por último, gracias por consultar esta documentación auxiliar a “Rulando”, y también espero que le haya servido de ayuda para comprender un poco más el funcionamiento de dicho programa.

**Referencias bibliográficas**

1. S. Russel, P. Norvig. Inteligencia Artificial: un enfoque moderno, Capítulo 10: Planning, Prentice-Hall, 3ª ed. (2010).
2. José Luis Ruiz Reina. Tema 8: Planificación Inteligencia Artificial I, 2012
3. Pedro Garduño Carmona. Tesis: Sistemas de planeación basados en el enfoque de aútomatas finitos.
4. Wikipedia. Formalismo PDDL.
5. A. Márquez, C. Del Valle, R. M. Gasca, M. Toro, Especificación PDDL de un Dominio de Ensamblaje.
6. Gerardo Parra, Sobre la Revisión de Planes en Agentes Inteligentes.