Planificación de Orden Parcial: El Mundo de los Bloques

Abner Guzmán, *Student Member, IEEE,* José Ilitzky, Cesar Montero, Guillermo Hernández y Erik Yñigo

***Resumen*—El mundo de los bloques es utilizado como marco de desarrollo y experimentación de un planificador de orden parcial. Se realizan pruebas en un mundo físico con bloques reales y un manipulador serial. El planificador obtenido resuelve eficientemente instancias del problema con decenas de bloques.**

***Índices*—Planificación, planificador de orden parcial, planificador no lineal, POP, mundo de los bloques, inteligencia artificial.**

I. Introducción

El problema del mundo de los bloques es clásico en la Inteligencia Artificial. Aunque es relativamente sencillo, permite experimentar con las técnicas de planificación y es una herramienta útil en la investigación y desarrollo de planificadores.

El mundo de los bloques consiste en un conjunto de bloques dispuestos sobre una mesa. Los bloques pueden ser apilardos, pero sólo uno puede descansar directamente sobre otro. Un manipulador puede levantar un bloque a la vez y posicionarlo en otra posición, ya sea encima de la mesa o de otro bloque. El objetivo es encontrar los pasos a seguir para llegar de una configuración o estado inicial de pilas a un estado final deseado.

Planificación se refiere a la elaboración de la lista de pasos a seguir para llegar a la solución. Cuando un planificador trabaja en todo momento con una lista de pasos ordenados, se le llama *planificador de orden total*. Sin embargo, esto resulta limitante e ineficiente. Si el planificador puede agregar pasos sin especificar siempre un orden entonces se le conoce como *planificador de orden parcial*. De esta forma, se especifican relaciones entre conjuntos de pasos, por ejemplo: $p_1 < p_2$ indica que el paso – o los pasos – p_1 aparecerá en el plan antes que p_2, pero no necesariamente inmediatamente antes.

En este artículo se describe la implementación de un planificador de orden parcial que puede resolver con eficiencia diversas instancias del problema del mundo de los bloques. Se reportan resultados de experimentos con una PC, un manipulador serial y bloques de madera. También se proponen ciertas mejoras o extensiones al proyecto.

A. Guzmán está en ITESM, Campus Guadalajara, Guadalajara, Jalisco. México (e-mail: abnerguzman@ieee.org).

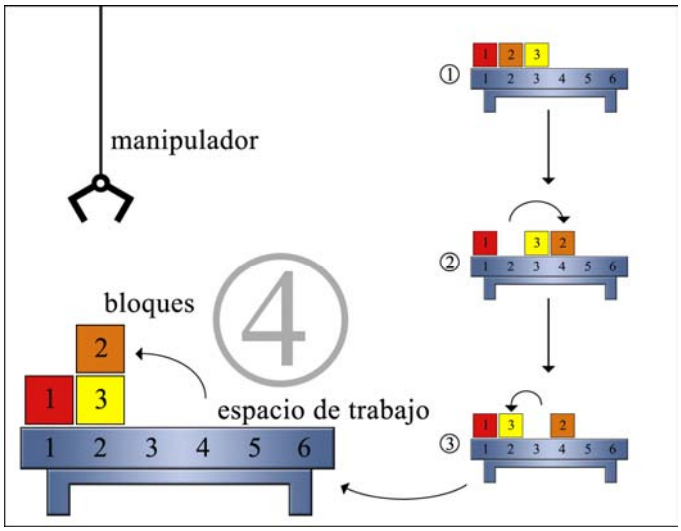

Fig. 1. Esquema del mundo de los bloques.

II. Trabajos Relacionados

Las ideas fundamentales en la planificación de orden parcial incluyen la detección de conflictos [16] y la protección de las condiciones ya logradas de posibles interferencias. La construcción de planificadores parciales fue iniciada en la segunda mitad de la década de los 1970's por [13], [14] con el planificador NOAH y por [17], [18] con el sistema NONLIN.

Los planificadores de orden parcial dominaron los siguientes 20 años en las investigaciones, aunque por mucho tiempo este campo no fue completamente entendido. Sin embargo, trabajos como TWEAK de [4], que consistía en reconstrucciones y simplificaciones lógicas de planificaciones, ayudaron a su comprensión. Poco tiempo después, una implementación del algoritmo de McAllester y Rosenblitt llamada SNLP [15] fue ampliamente distribuida y permitió a muchos investigadores entender y experimentar con planificadores de orden parcial por primera vez.

Alrededor de los 1990's la popularidad de los planificadores de orden parcial fue disminuyendo a medida que métodos más rápidos fueron emergiendo. En 2001, [11] rehabilita los métodos parciales, y con funciones heurísticas precisas derivadas de un grafo de planificación se logra agilizar y hacer más eficientes estos planificadores a tal grado que fueran competitivos con los planificadores de espacio-de-

estados más rápidos.

Tecnologías alternas como la planificación planteada como satisfactibilidad y el algoritmo SATplan, fueron propuestas por [7], siendo inspirados por el éxito de la búsqueda avara local para problemas de satisfactibilidad. [8] también investiga varias formas de representación proposicional para axiomas de STRIPS, concluyendo que las formas más compactas no siempre acarreaban tiempos de solución más rápidos.

Por otro lado, el resurgimiento del interés en la planificación de espacio-de-estados fue llevada a cabo por el trabajo de [9] y su programa UnPOP. En este, se sugería una función heurística de distancia basada en un problema relajado e ignorando las listas de borrado. Partiendo de este punto [3] desarrolla un planificador de búsqueda heurística que hizo de la búsqueda en espacio-de-estados una solución práctica para problemas de planificación grandes.

Con tal variedad de tecnologías, surgen comparaciones interesantes entre los diversos enfoques. [5] demuestra que los enfoques basados en limitantes (constraints), tales como GRAPHPLAN y SATplan, son los mejores para dominios NP-Duros, mientras que enfoques basados en búsquedas son mejores en dominios donde soluciones admisibles son encontradas sin la necesidad de aplicar retroceso (backtracking).

[1] y [2] toman en cuenta que los *algoritmos de planificación clásicos* ignoran las diferentes limitantes que existen cuando pasamos de una simulación virtual del mundo de los bloques a una simulación real o más aún a una aplicación real. Lo que ellos proponen es entonces traducir dichas limitantes a *metas clásicas* que un algoritmo clásico pueda manejar.

[6] demuestra que todo planificador parcial es una instancia de un algoritmo genérico ahí propuesto. Las diferentes decisiones tomadas en el diseño de un planificador parcial corresponden a diferentes instancias de ese algoritmo genérico. Su trabajo proporciona un análisis de los efectos que las decisiones en el diseño acarrean sobre la eficiencia y sobre el espacio de búsqueda del planificador resultante.

III. Metodología

Generalmente los problemas de planificación se definen en términos de *metas últimas (clásicas)*: las condiciones que deben sostenerse en el *estado final* luego de la ejecución del plan. Sin embargo en la práctica este tipo de planes pueden resultar insatisfactorios ya que los usuarios reales tienen por lo general otro tipo de requerimientos que deben atenderse, *metas no últimas* [2].

En el caso específico de este proyecto, las *metas últimas* – por ejemplo, el bloque 1 debe estar encima del bloque 2 – están acompañadas de restricciones relacionadas con el espacio de trabajo del robot – por ejemplo, no puede haber en ningún momento más de *n* bloques sobre la mesa. El objetivo es obtener un plan que el robot pueda llevar a cabo.

No obstante, el interés principal era el de conseguir un planificador clásico (un planificador que atiende *metas últimas*) en primera instancia y no un planificador *modificado*. Esto con la intención de conservar en lo posible la generalidad de la implementación y teniendo en mente alternativas como las propuestas en [2] para el logro de *metas no últimas*.

Por lo anterior la generación del *plan último* (el plan que será ejecutado por el robot) se da en dos etapas claramente delimitadas:

A. Generación de un plan por medio de un planificador clásico.
B. Refinamiento del plan y aplicación de restricciones.

Estas etapas se describen con detalle a continuación y en seguida, la sección C ilustra mediante un ejemplo los conceptos introducidos.

A. Generación de un plan por medio de un planificador clásico

El planificador implementado es un planificador de orden parcial con las siguientes características:

1) Política de elección de sub-metas:

Ya que el problema del mundo de los bloques es en esencia un problema de apilar bloques, resulta natural elegir primero *metas últimas* (en adelante, metas) de *abajo hacia arriba*. De esta forma podemos decir que el planificador trabaja por *niveles*.

Un *nivel* se define como un conjunto de metas que definen la posición de bloques que se encuentran a una misma *altura* (en el estado meta) o de manera equivalente, bloques que tienen exactamente el mismo número de bloques por debajo de ellos.

El planificador comienza tratando de satisfacer metas en el *primer nivel* – el nivel de los bloques que se encuentran sobre la mesa – y continúa de forma iterativa con los niveles siguientes sólo hasta que ha considerado todas las metas del nivel *actual*.

Dentro del nivel *actual* la elección de las metas no se encuentra predispuesta.

2) Operadores:

Todos los operadores que conforman el plan son del mismo tipo. La siguiente es una descripción del tipo de operador utilizado (como regla de STRIPS [10]):

move(x, y, z)
pc: on(x,y)^clear(x)^clear(z)
d: clear(z), on(x,y)
a: on(x,z), clear(y)

El operador *move* es instanciado al vincular las variables *x, y, z* con algún bloque en específico – una constante. Sin embargo, la instanciación no debe ser forzosamente total – pueden tenerse variables no vinculadas.

El operador *move* permite al planificador hacer uso del *principio del mínimo compromiso*: “si existe incertidumbre con respecto a la correctez de un compromiso, comprometa lo menos posible”, [19]. Aunque el nombre no lo sugiere, es

evidente que algún grado de compromiso es indispensable.

Este planificador siempre que agrega un operador al plan actual vincula al menos dos de las tres variables y siempre vincula *x*.

3) Reordenamiento de operadores:

La implementación del planificador busca minimizar la necesidad de reordenar operadores. Esto se logra como consecuencia de la manera en que se seleccionan las *sub-metas* y del agrupamiento de operadores en torno a estas – la generación de estos *grupos* es fundamental para el reordenamiento –, entre otras cosas.

La necesidad de realizar reordenamientos se puede dar como segundo paso a las unificaciones de operadores. Cuando el planificador determina que un par de operadores son unificables debe tomar en cuenta relaciones que se dan entre los grupos de operadores a los que los operadores a unificar pertenecen.

El principal interés al realizar un reordenamiento es el de dar lugar a un sub-plan (un plan que alcanza algunas *sub-metas*) ordenado (un plan en el que los operadores no tienen conflictos entre sí, ni conflictos con sus precondiciones).

4) Linearización del plan:

La última acción realizada por el planificador es la generación de un *plan lineal*. Las mismas estrategias que permitieron reducir los reordenamientos durante la planificación hacen que esta etapa sea muy transparente. Consiste simplemente en recorrer el grafo de operadores en el orden correcto y agregar los operadores a la lista final.

El plan generado por esta etapa puede o no ser *completo* en el sentido de que como consecuencia del principio de mínimo compromiso es probable que existan aún operadores parcialmente instanciados.

B. Refinamiento del plan y aplicación de restricciones

Esta segunda etapa busca hacer del *plan lineal*, generado por la etapa precedente, un *plan completo* y además realizable dadas las consideraciones de espacio.

Si en el plan existen aún operadores con variables no vinculadas deberá hacerse la vinculación en esta etapa. Podemos describirla en términos de tres acciones paralelas.

1) Vincular variables, caso move(A, B, z):

Un operador de esta forma indica que el planificador determinó necesario mover el bloque A para liberar al bloque B. Además a esa altura del plan el bloque A no puede ser movido a su posición final por lo que deberá moverse a alguna posición *temporal*.

La elección de esta posición temporal se dirige por los siguientes criterios:

i. Tratar de bloquear el menor número de movimientos subsecuentes.
ii. Preferir posiciones cercanas.

2) Vincular variables, caso move(A, x, C):

Este operador simplemente indica que en la etapa precedente este operador se generó después de haberse generado un operador del tipo move(A, B, z).

Es suficiente entonces vincular x con la reciente vinculación de z.

3) Evitar conflictos:

La vinculación de variables puede resultar en *bloqueos*, por ejemplo: move(A, B, C) bloquearía a move(C, D, E). Si en esta segunda etapa se detecta un caso análogo deberá resolverse de alguna forma, existen dos posibilidades:

A. Reordenar.
B. Agregar un nuevo operador.

El reordenamiento no es siempre una solución viable. En la implementación actual se optó por ii.

C. Ejemplo: generación de un plan simple

La descripción de una configuración o estado del mundo de los bloques puede hacerse por medio de un conjunto de relaciones de tipo:

on(x,y)

Esta relación indica que el bloque *x* esta ubicado en la posición *y* que puede ser encima de otro bloque o en una localidad sobre la mesa.

En este contexto utilizaremos las siguientes notaciones:

Bi, el bloque i.
Tj, la posición j sobre la mesa.
Ok, el k-ésimo operador agregado al plan.

Además, las transformaciones del plan se mostrarán en negritas.

Nuestro ejemplo consiste en generar un plan completo que permita llevar al mundo de los bloques del estado *S1* al estado *S2*, Fig. 2.:

S1 = {on(B1,T1), on(B4,T3), on(B7,T4), on(B2,B1), on(B5,B4), on(B8,B7), on(B3,B2), on(B6,B5)}

S2 = {on(B1,T2), on(B4,T3), on(B7,T4), on(B3,B1), on(B2,B4), on(B8,B7), on(B6,B8), on(B5,B6)}

Las relaciones presentes en *S2* representan *metas últimas*, cuando las consideramos de forma independiente, *sub-metas*.

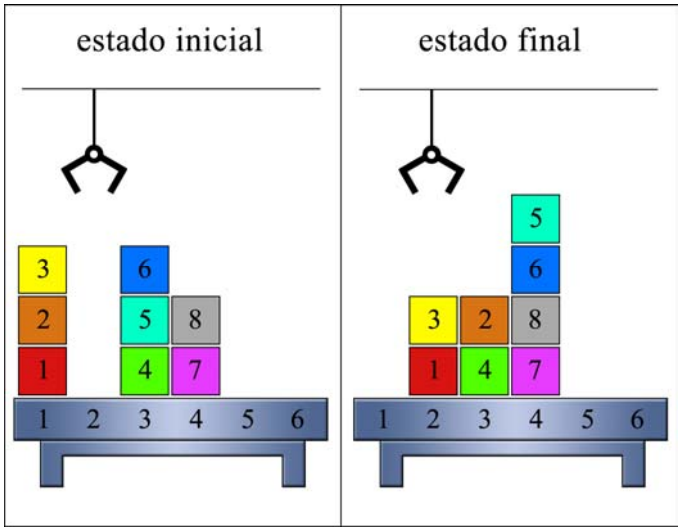

Fig. 2. Instancia del problema del mundo de los bloques.

El planificador inicia con un plan *P* (conjunto de

operadores) vacío y elige una *sub-meta* en el *primer nivel*, digamos:

on(B1,T2)

Esto da lugar a un *grupo* de operadores. El plan resultante es:

O1: move(B3,B2,z)
O2: move(B2,B1,z)
O3: move(B1,T1,T2)

Este primer paso ilustra la instanciación parcial de operadores. La variable *z* no es vinculada para *O1* ni para *O2*.

La siguiente *sub-meta* forma parte del *segundo nivel*:

on(B3,B1)

El planificador instancia un nuevo operador y lo agrega al plan cuidando evitar cualquier conflicto:

O1: move(B3,B2,z)
O2: move(B2,B1,z)
O3: move(B1,T1,T2)
O4: move(B3,y,B1)

Observamos que el nuevo operador, *O4*, se encuentra también parcialmente instanciado debido a que no es posible determinar la posición de *B3* como consecuencia de *O1*.

El planificador continua con la *sub-meta*:

on(B2,B4)

La satisfacción de ésta puede segmentarse en dos pasos, liberar a *B4* y mover a *B2* a su posición final. El planificador transforma el plan con el siguiente resultado:

O1: move(B3,B2,z)
O5: move(B6,B5,z)
O6: move(B5,B4,z)
*O2: move(B2,B1,**B4**)*
O3: move(B1,T1,T2)
O4: move(B3,y,B1)

La liberación de *B4* se consigue agregando dos nuevos operadores – una vez más cuidando evitar conflictos. El movimiento de *B2* ilustra el poder de la *unificación*. En lugar de agregar un tercer operador, el planificador fue capaz de determinar que el previo *O2* puede ser utilizado para conseguir la *sub-meta* – el movimiento correcto en el momento preciso. Vinculando *z* con *B4*, *O2* es totalmente instanciado.

Las siguientes dos metas se encuentran ya en el *tercer nivel*, por su simplicidad y similitud se tratan aquí en conjunto:

on(B6,B8) y on(B5,B6)

De nueva cuenta la *unificación* resulta ser la solución:

O1: move(B3,B2,z)
*O5: move(B6,B5,**B8**)*
*O6: move(B5,B4,**B6**)*
O2: move(B2,B1,B4)
O3: move(B1,T1,T2)
O4: move(B3,y,B1)

En este momento todas las *metas últimas* han sido consideradas y termina la *primera etapa* del algoritmo. Sin embargo el plan aunque *lineal*, es aún incompleto ya que existen dos operadores parcialmente instanciados, *O1* y *O4*. Ambos operadores tienen que ver con un bloque que debe pasar un tiempo en una *posición temporal*. La *segunda etapa* del algoritmo selecciona la posición *T5* para este fin y con esto termina de instanciar los operadores. El plan resultante es completo y realizable en nuestro mundo físico:

*O1: move(B3,B2,**T5**)*
O5: move(B6,B5,B8)
O6: move(B5,B4,B6)
O2: move(B2,B1,B4)
O3: move(B1,T1,T2)
*O4: move(B3,**T5**,B1)*

IV. Resultados Experimentales

El entorno utilizado para la experimentación se muestra en la Fig. 3. Consiste en un conjunto de bloques de madera dispuestos en posiciones bien conocidas dentro del área de trabajo del manipulador. El manipulador utilizado es un robot Mitsubishi Movemaster EX con 5 grados de libertad.

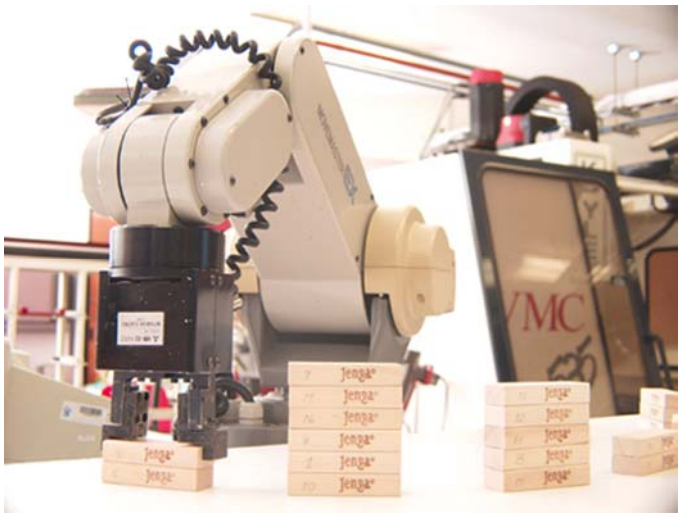

Fig. 3. El mundo de los bloques.

Las restricciones de espacio fueron:

1. Se pueden colocar como máximo 6 bloques directamente sobre la mesa.
2. La pila de altura máxima esta conformada por 6 bloques.

El manipulador fue controlado por medio de un programa en código-G generado a partir de la lista de operadores (el plan completo). Cada operador move(x, y, z) fue traducido en 8 movimientos (líneas de código-G) – el menor número posible, en el caso general, para evitar colisiones.

La implementación del algoritmo se hizo utilizando programación orientada a objetos y se utilizó el lenguaje de programación Java. Las corridas se hicieron en una computadora con procesador Pentium III a 1 GHz y 512 KB de memoria principal. Para las instancias del problema probadas se generaron planes en tiempos de alrededor de 200 ms.

Entre los diversos objetos construidos destacan los

siguientes:

1. State: representa un estado o configuración del mundo de los bloques.
2. MoveSchema: representa un operador total o parcialmente instanciado.
3. Planner: interfase que implementa todo planificador en nuestra arquitectura.
4. POPlanner: implementación no lineal de Planner.

POPlanner representa la realización de la mayor parte del trabajo presentado en este artículo – el planificador clásico o primera etapa. Este objeto se vale de un grafo especializado para construir el plan. Las relaciones entre operadores son representadas de forma implícita en la estructura del grafo. Además, todas las transformaciones al plan se realizan sobre el mismo. POPlanner hace uso extensivo de tablas hash para agilizar la búsqueda de operadores dentro del grafo – por ejemplo, al intentar alguna unificación.

Existen otras clases relacionadas con las siguientes áreas dentro del proyecto:

1. Representación del grafo de planificación.
2. Traducción del plan completo a un programa en código-G del manipulador utilizado en las pruebas física.
3. Comunicación (serial) con el manipulador.
4. Interfase de usuario (gui) para la fácil descripción de estados de interés y el control general de los experimentos.

La Fig. 4 muestra en un diagrama los distintos componentes que interactuaron en los experimentos.

El planificador obtenido puede generar planes correctos para resolver prácticamente cualquier instancia del problema del mundo de los bloques cuando las restricciones de espacio se relajan.

Siempre que el planificador pudo completar un plan, también le fue posible al manipulador llevarlo a cabo con los resultados esperados. No obstante en condiciones de *espacio reducido* – es decir cuando todas las pilas están llenas o casi llenas – el planificador (segunda etapa) se vio forzado a abortar la planificación. Algunas de estas *instancias saturadas* son realmente imposibles de resolver dadas las restricciones de espacio, pero algunas otras no lo son.

Este hecho es consecuencia de la segmentación que se hizo de la planificación. Más concretamente es resultado inmediato del hecho de que la segunda etapa de planificación no contempla reordenamientos, estos pueden llegar a ser indispensables en este tipo de instancias del problema.

V. Conclusiones y Trabajo a Futuro

El objetivo central del proyecto fue alcanzado satisfactoriamente, el planificador resuelve eficientemente instancias del problema del mundo de los bloques con decenas de bloques. Sin embargo, la segmentación del planificador demostró ser limitante. Es deseable que el planificador pueda generar el plan lineal, completo y último en una sola etapa.

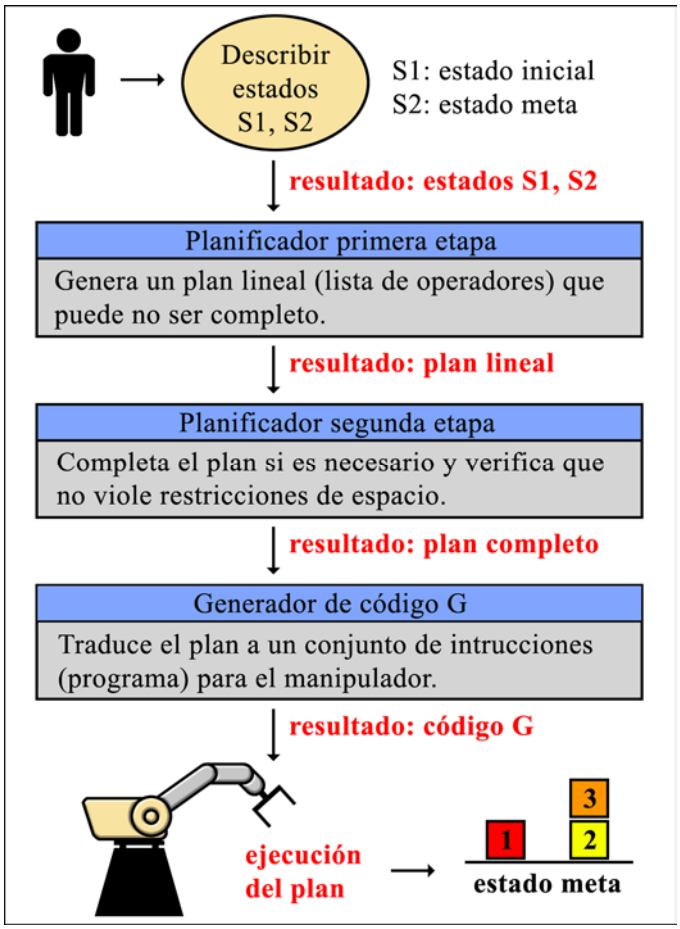

Fig. 4. Interacción de componentes en los experimentos.

Técnicas similares a las desarrolladas en [2] podrían ser aplicadas y permitir que el planificador actual (primer etapa) sea extendido para conseguir lo anterior.

Una manera más de extender el proyecto será la incorporación de técnicas de visión computacional. Habilitar al agente planificador con la capacidad de determinar la configuración actual de los bloques en todo momento para guiar sus movimientos en el espacio y para evitar al usuario la necesidad de describir el estado inicial.

VI. Agradecimientos

Los autores agradecen las enseñanzas, contribuciones y motivación del Dr. Gildardo Sánchez Ante.

VII. Referencias

[1] M. Baioletti, S. Marcugini y A. Milani, "Partial Plans Completion with GRAPHPLAN", Department of Mathematics and Computer science, University of Perugia, Italy, 1998.

[2] M. Baioletti, S. Marcugini y A. Milani, "Encoding Planning Constraints into Partial Order Planning Domains", Department of Mathematics and Computer science, University of Perugia, Italy, 1998.

[3] B. Bonet y H. Geffner, “Planning as Heuristic Search: New Results”, Proceedings of the European Conference on Planning, pp. 360 – 372, Durham, UK. Springer-Verlag, 1999.

[4] D. Chapman, “Planning for conjuctive goals”, Artificial Intelligence, 32(3), pp. 333 – 377, 1987.

[5] M. Helmert, “On the Complexity of Planning in Transportation Domains”, Sixth European Conference on Planning (ECP-01), Toledo, Spain, Springer-Verlag, 2001.
[6] S. Kambhampati, “Design Tradeoffs in Partial Order (Plan Space) Planning”, Department of Computer Science and Engineering, Arizona State University, Tempe, Arizona, 1994.
[7] H. Kautz y B. Selman, “Planning as Satisfiability” en ECAI 92: 10th European Conference on Artificial Intelligence Proceedings, pp. 359 – 363, Vienna, Wiley, 1992.
[8] H. Kautz, D. A. McAllester y B. Selman, “Encoding Plans in Propositional Logic”, en Proceedings of the Fifth International Joint Conference on Principles of Knowledge Representation and Reasoning, pp. 374 – 384, Cambridge, Massachusetts. Morgan Kaufmann, 1996.
[9] D. McDermott, “A Heuristic Estimator for Means-Ends Analysis in Planning”, en Proceedings of the Third International Conference on AI Planning Systems, pp. 142 – 149, Edinburgh, Scotland. AAAI Press, 1996.
[10] N. Nilsson, "Artificial Intelligence: A New Synthesis", San Francisco, CA: Morgan Kauffman, 1998.
[11] X. Nguyen, y S. Kambhampati, “Reviving Partial Order Planning”, en Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence (IJCAI-01), pp. 459 – 466, Seattle. Morgan Kaufmann, 2001.
[12] S. Russell, y P. Norvig, "Artificial Intelligence: A Modern Approach 3rd ed", Englewood Cliffs, NJ: Prentice Hall, 2003.
[13] E. D. Sacerdoti, “The nonlinear nature of plans”, en Proceedings of the Fourth International Joint Conference on Artificial Intelligence (IJCAI-75), pp. 206 – 214, Tbilisi, Georgia. IJCAII, 1975.
[14] E. D. Sacerdoti, “A Structure for Plans and Behavior”, Elsevier/North-Holland, Amsterdam, London, New York, 1977.
[15] S. Soderland, y D. S. Weld, “Evaluating nonlinear planning”, Technical report TR-91-02-03, University of Washington Department of Computer Science and Engineering, Seattle, Washington, 1991.
[16] A. Tate, “Interactive goals and their use”, en Proceeding of the Fourth International Joint Conference on Artificial Intelligence (IJCAI-75), pp. 215 – 218, Tbilisi, Georgia. IJCAII, 1975.
[17] A. Tate, “Using Goal Structure to Direct Search in a Problem Solver”, Tesis doctoral, University of Edinburgh, Edinburgh, Scotland, 1975.
[18] A. Tate, “Generating Project Networks”, en Proceedings of the Fifth International Joint conference on Artificial Intelligence (IJCAI-77), pp. 888-893, Cambridge, Massachusetts. IJCAII, 1977.
[19] P. H. Winston, "Artificial Intelligence 3rd ed", Addison-Wesley, 1992.

VIII. BIOGRAFÍAS

Abner Guzmán nació en Guadalajara, Jalisco, México el 24 de enero de 1982. Actualmente cursa la carrera de Ingeniería en Sistemas Electrónicos en el Instituto Tecnológico y de Estudios Superiores de Monterrey, Campus Guadalajara.

Sus áreas de interés incluyen la arquitectura de computadoras, la inteligencia artificial, el diseño de algoritmos, el control automático y el diseño electrónico.

Es miembro del IEEE desde 2004.

José Ilitzky nació en Guadalajara, Jalisco, México el 7 de septiembre de 1982. Estudia la carrera de Ingeniería en Sistemas Computacionales en el Instituto Tecnológico y de Estudios Superiores de Monterrey, Campus Guadalajara.

Su experiencia laboral incluye el desarrollo de soluciones en Web para las empresas Hermes Music, Mirars, Calquetza, CITE e Innox. Sus campos especiales de interés incluyen las graficas computacionales, la inteligencia artificial y la robótica.

Cesar Montero nació en Guadalajara, Jalisco, México el 3 de noviembre de 1981. Estudia la carrera de Ingeniería en Sistemas Computacionales en el Instituto Tecnológico y de Estudios Superiores de Monterrey, Campus Guadalajara.

Su experiencia laboral incluye el Centro de Innovación Tecnológica Educativa, Zigno, la Compañía Portal Siete, Republica Interactive y Accend Consulting. Sus campos especiales de interés incluyen el diseño, desarrollo y programación de sistemas relacionados con la animación.

Guillermo Hernández nació en Guadalajara, Jalisco, México el 24 de diciembrre de 1981. Estudia la carrera de Ingeniería en Sistemas Computacionales en el Instituto Tecnológico y de Estudios Superiores de Monterrey, Campus Guadalajara.

Sus campos especiales de interés incluyen: animación 3D, desarrollo de software, multimedia y la música.

Erik Yñigo nació en Cd. Obregón, Sonora, México el 29 de mayo de 1982. Actualmente continúa sus estudios de Ingeniería en Sistemas Computacionales en el Instituto Tecnológico y de Estudios Superiores de Monterrey, Campus Guadalajara.

Su experiencia laboral incluye el Centro de Inovación y Tecnología Educativa y la compañía de software multiplataforma Innox. Sus campos especiales de interés incluyen la programación, el diseño de interfaces, usabilidad, diseño gráfico, animación y desarrollo de multimedia.