UNIVERSIDAD SIMÓN BOLÍVAR

Ingeniería de la Computación

Diseño automático de heurísticas basadas en bases de datos de patrones para planificación óptima

Por

Eduardo Marcelo Feo Flushing

PROYECTO DE GRADO
Presentado ante la Ilustre Universidad Simón Bolívar
como Requisito Parcial para Optar al Título de
Ingeniero en Computación

Sartenejas, Marzo de 2007

UNIVERSIDAD SIMÓN BOLÍVAR

DECANATO DE ESTUDIOS PROFESIONALES

COORDINACIÓN DE INGENIERÍA DE LA COMPUTACIÓN

ACTA FINAL DEL PROYECTO DE GRADO

Diseño automático de heurísticas basadas en bases de datos de patrones para planificación óptima

Presentado por:

Eduardo Marcelo Feo Flushing

Este proyecto de grado ha sido aprobado por el siguiente jurado examinador:

_____

Prof. Alexandra La Cruz (Jurado)

_____

Prof. Ivette Carolina Martínez (Jurado)

_____

Prof. Blai Bonet (Tutor)

Sartenejas, Marzo de 2007

Diseño automático de heurísticas basadas en bases de datos de patrones para planificación óptima

Por

Eduardo Marcelo Feo Flushing

Uno de los objetivos de la Inteligencia Artificial desde sus inicios ha sido el desarrollar métodos y técnicas para resolver, de forma automática, problemas que están definidos a un alto nivel. A partir de allí surge la idea de planificación. En este proyecto nos enfocamos en dos conceptos: la planificación clásica, la cual se refiere a problemas de decisión secuencial, en un espacio determinístico, y la independencia del dominio, que se refiere a desarrollar solucionadores generales de problemas.

Uno de los enfoques para hacerle frente a los problemas de planificación clásica es la planificación como búsqueda heurística. Los planificadores que aplican este enfoque transforman el problema de planificación en un problema de búsqueda y, debido a la independencia de dominio, deben extraer heurísticas de la definición del problema para controlar y guiar de forma efectiva la búsqueda. Las heurísticas, funciones que estiman la distancia en el espacio de búsqueda, son una técnica muy importante para ejerce este control. Para garantizar la optimalidad, que es uno de los objetivos del proyecto, es necesario que estas heurísticas sean admisibles, es decir, no sobreestimen la distancia en el espacio de búsqueda.

Recientemente, se han utilizado las bases de datos de patrones para derivar heurísticas admisibles. Una base de datos de patrones (PDB) es una función heurística, basada en memoria, obtenida al considerar sólo una parte del problema (el patrón) lo suficientemente pequeña como para ser resuelta para todo estado mediante una búsqueda exhaustiva. La calidad de la heurística aumenta a medida que se incrementa el tamaño del patrón. Sin embargo, el principal problema de este tipo de heurísticas es que la cantidad de tiempo y memoria necesaria para su cálculo limitan el tamaño del patrón. Es por ello que la calidad de la heurística depende de forma crucial de la selección de los patrones. Este proyecto plantea nuevos métodos de selección de patrones con la finalidad de obtener mejores heurísticas admisibles basadas en bases de datos de patrones.

Los experimentos demostraron que uno de los dos métodos propuestos es competitivo con los ya existentes. Considerando que el determinismo de los métodos es importante debido a que implica menos ambigüedad durante la selección, el nuevo método demostró ser más determinístico. Tomando en cuenta el tiempo de cálculo de la heurística, utilizando cada uno de los métodos, se pudo concluir que el tiempo que consume el proceso de selección no es relevante tomando en cuenta el tiempo de elaboración de las tablas. El análisis y las conclusiones señalan varias áreas en las cuales pueden realizarse futuros estudios.

ÍNDICE GENERAL

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

Lista de Algoritmos

Capítulo 1

INTRODUCCIÓN

Planificación en Inteligencia Artificial puede definirse de forma general como el arte de resolver problemas. La idea principal es desarrollar métodos y técnicas para resolver, de forma automática, problemas que están definidos a un alto nivel. Este ha sido uno de los objetivos principales de la Inteligencia Artificial desde sus inicios [13]. Este estudio estará enmarcado en varios conceptos. El primero de ellos, planificación clásica, refiere a la resolución óptima de problemas de decisión secuencial, en un espacio determinístico, donde se posee toda la información del dominio. El segundo, independencia de dominio, consiste en el desarrollo de planificadores para cualquier problema de planificación en general.
Actualmente existen diversos enfoques para hacerle frente a la planificación. Algunos de los más renombrados son *Graphplan*, basado en ideas utilizadas en algoritmos de grafos [4], y *Satplan*, basado en satisfacibilidad de fórmulas proposicionales [23]. En este proyecto nos centraremos en uno de ellos, planificación como búsqueda heurística, el cual ha sido estudiado ampliamente por Bonet y Geffner [7, 8] y otros investigadores [29, 16]. Los planificadores que aplican este enfoque transforman el problema de planificación en un problema de búsqueda y, por ser independientes del dominio, deben extraer heurísticas de la definición del problema para guiar de forma efectiva la búsqueda. El objetivo de este proyecto es contribuir al desarrollo de métodos para elaborar estas heurísticas.

Heurísticas admisibles en planificación

Lograr la optimalidad de la solución puede llegar a ser muy complicado en muchos problemas de planificación. El efectivo control de la búsqueda es imprescindible y las heurísticas, funciones que estiman la distancia en el espacio de búsqueda, son una técnica muy importante para ejercer este control [16]. Para lograr soluciones óptimas, es necesario que estas heurísticas sean admisibles, es decir, que no sobreestimen la distancia real. Dado que en la planificación independiente del dominio sólo se dispone de la descripción del problema, cualquier método debe valerse únicamente de ésta. Es por ello que surge la

idea y la necesidad de derivar heurísticas a través de un análisis del problema.

La flexibilidad y la eficiencia son los principales retos en la planificación como búsqueda heurística [7], de éstos, la eficiencia puede ser lograda mediante la formulación, análisis y desarrollo de heurísticas y optimizaciones. A lo largo de los años, diferentes tipos de heurísticas han sido formuladas y estudiadas a profundidad, la mayoría de las cuales se basan en realizar relajaciones al problema [7, 4, 31]. Un modelo de relajación utilizado recientemente para derivar heurísticas admisibles está basado en la idea de bases de datos de patrones. Las bases de datos de patrones han sido utilizadas para la resolución de problemas como el 24-puzzle [26] y el cubo de Rubik [25]. Una base de datos de patrones (PDB) es una función heurística, basada en memoria, obtenida al considerar sólo una parte del problema (el patrón) lo suficientemente pequeña como para ser resuelta para todo estado mediante una búsqueda exhaustiva. La aplicación de las bases de datos de patrones en la planificación como búsqueda heurística fue introducida por Edelkamp [10]. Aplicadas a la planificación STRIPS, los patrones son considerados como un subconjunto de átomos del problema, ignorando los demás átomos. El patrón, definido como un subconjunto de átomos A, define un problema de planificación simplificado, el cual llamaremos problema abstracto P^A, donde los estados y las listas asociadas a los operadores son intersectadas con A [16]. Definimos la heurística admisible h^A, la cual se calcula resolviendo el problema abstracto completamente, a través de una búsqueda exhaustiva. El valor de la solución óptima en el problema abstracto P^A es una cota inferior del valor de la solución para el problema original P.

Edelkamp [10] propone, para optimizar el uso de la memoria y la consulta de la PDB, utilizar una representación en variables multi-valuadas, que están definidas implícitamente en el problema. En este proyecto fue implementada una representación de variables, llamada SAS. Tales variables corresponden a un tipo de invariante particular [17]. Bajo la nueva representación, los patrones corresponden a un conjunto de variables. La calidad de la heurística puede mejorar a medida que se consideran patrones de mayor tamaño. Sin embargo, el principal problema de las heurísticas PDB es que el tamaño en memoria y la cantidad de tiempo necesaria para la construcción de las tablas limitan el tamaño del patrón. Por ello, la calidad de la heurística depende de forma crucial en la selección de los patrones. En la planificación independiente del dominio, donde el patrón apropiado varía entre cada problema, la selección debe ser automática, lo cual puede llegar a ser bastante problemático. Dado este problema, Bonet, Geffner y Haslum [17], introducen dos elementos de gran importancia para las heurísticas PDB. El primero, las bases de datos con restricciones, donde las restricciones del problema original son aplicadas a los problemas abstracto, y un método de selección de patrones, la *selección iterativa incremental*, ambos con la finalidad de mejorar la calidad de las heurísticas. El objetivo de este proyecto es el estudio y el planteamiento de nuevos métodos de selección de patrones para obtener mejores heurísticas admisibles

basadas en bases de datos de patrones.

1.1. Objetivos y Conclusiones

1.1.1. Objetivo General

El objetivo principal del proyecto es estudiar las heurísticas de bases de datos de patrones en la planificación clásica independiente del dominio. El estudio está enfocado en las estrategias de selección de patrones. Como base, se tomó la publicación de Bonet, Geffner y Haslum [17], donde se estudian las bases de datos de patrones en la planificación STRIPS y mencionan algunos aspectos relevantes en la construcción de las PDBS. El principal resultado de este estudio son nuevas estrategias para la selección de patrones, las cuales serán comparadas con la propuesta de Bonet, Geffner y Haslum [17] en diversos dominios.

Objetivos Específicos

- Implementación de un planificador STRIPS. En este punto se incluye: un parser de archivos *PDDL*, construcción del modelo de búsqueda, implementación de algoritmos para calcular las heurísticas de relajación (h^1, h^2), algoritmos de búsqueda (A*, IDA*), entre otros procesos necesarios para llevar a cabo la planificación.
- Implementación de heurísticas de bases de datos de patrones. En este punto podemos incluir la implementación de variables multi-valuadas, traducción del modelo STRIPS al modelo SAS, obtención de conjuntos aditivos de variables, implementación de bases de datos con restricciones y la estrategia de selección iterativa incremental de patrones.
- Investigación e implementación de nuevas estrategias de selección de patrones.
- Experimentación con diversos dominios para realizar una comparación entre las diferentes estrategias de selección de patrones.

1.1.2. Conclusiones

Se diseñaron e implementaron dos estrategias de selección de patrones. Una de ellas, el *Gap set*, demostró ser competitiva con la estrategia planteada por Bonet, Geffner y Haslum [17]. En cuanto a la eficiencia de los diferentes métodos para guiar la búsqueda, el *Gap set* mostró resultados uniformes al dominar en dos de los tres dominios considerados. Al realizar un análisis sobre el determinismo de los

métodos, considerando el porcentaje de empates a la hora de realizar la selección, el *Gap set* demostró ser el más determinístico. El determinismo en una estrategia de selección es importante dado que implica menos ambiguedad en el momento de seleccionar los patrones. En relación al tiempo de elaboración de cada una de las heurísticas utilizando los métodos estudiados, se pudo observar diferencias considerables entre cada uno de ellos y al mismo tiempo se pudo concluir que el tiempo que consume el proceso de selección no es determinante frente al tiempo de construcción de las tablas. El análisis de la calidad de las heurísticas obtenidas por los diferentes métodos de selección de patrones señalan varias áreas en las cuales pueden realizarse futuros estudios, entre las cuales se encuentra el desarrollo de nuevos métodos, o la implementación de mejoras a los ya existentes, para hacer las selecciones más informadas y más determinísticas.

1.2. Organización

En el siguiente capítulo explicaremos con más detalle los principales componentes de la planificación como búsqueda heurística. En el capítulo 3 estudiaremos los aspetos relacionados con la búsqueda heurística. En el capítulo 4 estudiaremos dos clases de heurísticas, la familia de heurísticas h^m y las bases de datos de patrones. En el capítulo 5 explicaremos detalles relevantes de la implementación del planificador, así como de las heurísticas. En los capítulos 6 y 7 explicaremos los experimentos realizados y las conclusiones de los mismos.

Capítulo 2

PLANIFICACIÓN

La planificación en Inteligencia Artificial comprende el desarrollo de *solucionadores generales de problemas*. Un solucionador general de problemas recibe la descripción de un problema en un lenguaje de alto nivel y de forma automática calcula su solución [13]. Existen varios tipos de planificación, entre los cuales podemos destacar la *planificación clásica* , la cual se basa en modelos de estados. Estos modelos comprenden acciones que representan transiciones determinísticas de un estado a otro. La solución en este tipo de planificación consiste en encontrar una secuencia de acciones que, aplicadas a partir de un estado inicial, nos permitan llegar a un estado objetivo. Entre otros tipos de planificación se destacan:

- La *planificación conformante*, donde las acciones son *no-determinísticas*.
- La *planificación temporal*, donde cada acción tiene una duración y la ejecución de algunas puede ocurrir de forma simultánea.

A lo largo de los años, han surgido muchos avances en la resolución de problemas de planificación clásica, la mayoría de los cuales se basan en las ideas propuestas por *Graphplan* [4], *Satplan* [23] y *HSP* [7]. *Graphplan*, un planificador introducido por [4] está basado en algoritmos de grafos. *Satplan* transforma el problema de planificación en una fórmula proposicional, sobre la cual se aplican algoritmos de satisfacibilidad para hallar una solución [23]. *HSP* [7] introdujo, por primera vez, la búsqueda heurística a las competencias de planificación. Los planificadores de búsqueda heurística, como *HSP*, transforman los problemas de planificación en problemas de búsqueda heurística, extrayendo automáticamente heurísticas de la definición del problema.
Las ideas propuestas y discutidas en este proyecto estarán enmarcadas en el concepto de planificación clásica, específicamente en la planificación como búsqueda heurística.

2.1. Planificación clásica

En la Inteligencia Artificial clásica, la planificación se aborda como un sistema de transición de estados, específicamente, utilizando *modelos de estados*. La resolución de los problemas en estos modelos se reduce a la obtención de un conjunto de transiciones que aplicadas de una forma particular a partir de un estado inicial, se alcanza un estado final, llamado objetivo. Este conjunto de transiciones es denominado *plan*, y puede ser óptimo en base a algún criterio, generalmente la suma de los costos de las transiciones. De esta definición simplificada podemos deducir las propiedades de un modelo de estados:

- Una noción de estados: donde se destacan un estado inicial y uno o varios estados objetivos.
- Un conjunto de transiciones entre estados, que a partir de ahora llamaremos acciones. Éstas pueden tener costos asociados.

Formalmente, un modelo de estados es una tupla $\langle S, s_0, S_G, A, f, c\rangle$ donde:

- S es un conjunto finito, no vacío, de estados.
- $s_0 \in S$ es el estado inicial
- $S_G \subseteq S$ es un subconjunto no vacío de estados objetivo
- $A(s) \subseteq A$ las acciones aplicables a cada estado s
- $f(a, s)$ denota una función de transición de estados para todo $s \in S$ y $a \in A(s)$
- $c(a, s)$ es el costo de aplicar la acción a en el estado s

Como se comentó anteriormente, un plan consiste en una secuencia de acciones $a_0, a_1, \ldots, a_n$ que a su vez definen una secuencia de estados $s_0, s_1 = f(a_0, s_0), \ldots, s_{n+1} = f(a_n, s_n)$ tal que cada acción a_i es aplicable para todo estado s_i y el último estado es objetivo, $s_{n+1} \in S_G$. El plan se considera óptimo si la suma de los costos es mínima.

2.1.1. STRIPS

El modelo que ha sido utilizado desde hace muchos años para describir los problemas de planificación es STRIPS, el cual fue introducido por Fikes y Nilsson en 1971 [11]. Un problema en STRIPS consiste en una tupla $\langle A, O, I, G\rangle$ donde A es un conjunto de átomos, O un conjunto de operadores y $I \subseteq A$,

$G \subseteq A$, la situación inicial y final respectivamente. Cada operador tiene asociado tres conjuntos de átomos: $Prec(op)$ $Add(op)$ y $Del(op)$, una precondición, un conjunto de átomos a agregar y un conjunto a eliminar. La representación STRIPS define un modelo de estados $S_P = \langle S, s_0, S_G, A(.), f, c\rangle$ donde:

- $s \in S$ son colecciones de átomos.
- $s_0 = I$ es el estado inicial.
- Los estados objetivo $s \in S_G$ son aquellos donde $G \subseteq s$.
- Las acciones $a \in A(s)$ son los operadores $op \in O$ tal que $Prec(op) \subseteq s$
- La función de transición f está definida como: $f(s, a) = s - Del(a) + Add(a)$ donde $a \in A(s)$
- Todos los costos de las acciones son iguales a 1.

Ejemplo: Blocksworld

El problema blocksworld es un ejemplo estandar de planificación [16]. Lo utilizaremos para ejemplificar el modelo STRIPS y el modelo de estados que éste deriva. Para este ejemplo utilizaremos la sintaxis PDDL, de la cual hablaremos más adelante.

Descripción del problema Se tiene un conjunto de bloques dispuestos sobre una mesa. Cada bloque tiene dos posiciones posibles: Estar reposando sobre la mesa directamente o sobre otro bloque. De forma análoga, cada bloque, puede o no, tener otro bloque encima. Las acciones que pueden realizarse son mover los bloques, siempre y cuando no interfieran otros. Como ejemplo, veamos la figura 2.1: A la izquierda, como disposición inicial, tenemos una pila tres bloques, los cuales denotaremos A, B y C, con el bloque C sobre la mesa, y el bloque A en el tope de la pila. La única acción posible en este estado es mover el bloque A del tope de la pila a la mesa. Los movimientos de los otros bloques están restringidos por la posición del bloque A. Ahora bien, en la figura a la derecha, luego de trasladar el bloque A a la mesa, tenemos tres acciones posibles: (1) Colocar a B sobre A, (2) trasladar el bloque A sobre B (lo cual nos regresaría al estado anterior) y (3) colocar el bloque B sobre la mesa.

Traducción a STRIPS (utilizando PDDL) Para representar el problema en STRIPS, lo primero que debemos hacer es modelar los estados como conjuntos de átomos. Los átomos que utilizaremos serán:

- `(on-table x)`: para indicar que el bloque x se encuentra sobre la mesa.
- `(on x y)`: para indicar que el bloque x se encuentra sobre el bloque y.

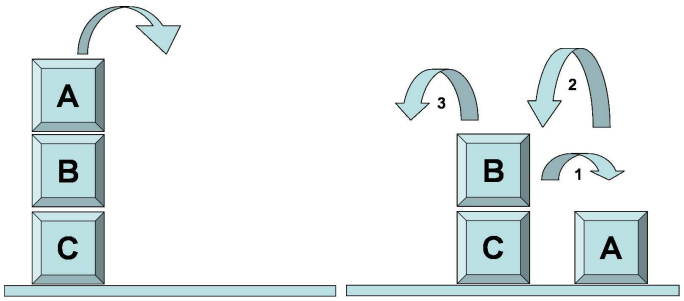


Figura 2.1: Ejemplo de blocksworld. Posibles movimientos de bloques en cada estado.

- `(clear x)`: señala que el bloque x no tiene ningún bloque sobre él.

Volviendo al ejemplo anterior, el estado inicial I estaría descrito por el conjunto:
$\{$`(on-table A)`, `(on B A)`, `(on C B)`, `(clear C)`$\}$.
Los operadores describen los movimientos posibles de los bloques, los cuales son:

- `(move-table x y)`: Traslada el bloque x, el cual se encuentra sobre y, a la mesa. Este operador es aplicable en aquellos estados donde el bloque x no tiene ningún bloque encima, por ende, la precondición es el conjunto $\{$`(clear x)`, `(on x y)`$\}$. El efecto del operador es *eliminar* el átomo `(on x y)` y *agregar* $\{$`(on-table x)`, `(clear y)`$\}$, por lo que éstas son sus listas *Del* y *Add* respectivamente.
- `(move-to-block x y)`: Coloca el bloque x, el cual se encuentra sobre la mesa, sobre el bloque y. La precondición: $\{$`(on-table x)`, `(clear x)`, `(clear y)`$\}$. Efecto: agrega $\{$`(on x y)`$\}$ y elimina $\{$`(on-table x)`, `(clear y)`$\}$.
- `(move-to-block x y z)`: Coloca el bloque x sobre el bloque z. En esta acción, x se encuentra sobre el bloque y, La precondición: $\{$`(on x y)`, `(clear x)(clear z)`$\}$. Efecto: agrega $\{$`(on x z)(clear y)`$\}$ y elimina $\{$`(on x y)`, `(clear z)`$\}$.

2.2. Planificación como búsqueda heurística

Para resolver un problema algorítmicamente, debe plantearse un modelo formal. Para resolver un problema mediante una búsqueda, el modelo formal debe ser traducido a un espacio de búsqueda. [16]. Una solución del modelo de estados es una solución para el problema STRIPS. Esto constituye la idea principal de la planificación como búsqueda heurística: Se traduce el modelo formal, en este caso STRIPS, a un modelo de estados, el cual es resuelto mediante una búsqueda, utilizando una heurística que es derivada automáticamente de la definición del problema [18]. Enfocándonos ahora en el problema de búsqueda, existen tres características principales: la dirección de la búsqueda, el algoritmo utilizado y la heurística derivada.

2.2.1. Dirección de búsqueda

La búsqueda para resolver un problema de planificación puede realizarse hacia adelante o hacia atrás. La *búsqueda hacia adelante* consiste simplemente en, a partir del estado inicial s_0, aplicar la función de transición f para generar los siguientes estados, así sucesivamente hasta hallar algún estado objetivo. La *búsqueda hacia atrás*, también conocida como *búsqueda en regresión*, tiene un enfoque más elaborado y es una idea trabajada desde hace muchos años en planificación [30, 33]. Bonet y Geffner [7] la proponen como una solución a la baja tasa de generación de nodos, debida principalmente al cálculo de la heurística en cada nuevo nodo generado. La búsqueda en regresión permite que algunas heurísticas puedan ser precalculadas y así agilizar la expansión de los nodos.

2.2.2. Algoritmos de búsqueda

En la planificación como búsqueda heurística, se ha utilizado muchos algoritmos [34, 7]. La selección del algoritmo ha utilizar está ligada al tipo de planificación: Óptima o sub-óptima. En general, el tipo de algoritmo no constituye un factor determinante para la resolución de un problema de planificación. Comparado con la calidad de la heurística, la selección del algoritmo de búsqueda no es un factor de importancia. En este trabajo, utilizamos A^* [30] debido a la simplicidad de implementación y su versatilidad. Detalles de implementación serán discutidos más adelante.

2.2.3. Heurísticas

La idea de derivar heurísticas de la definición del problema en planificación ha sido estudiada desde hace muchos años. La derivación de heurísticas se realiza generalmente en la planificación independiente del dominio, donde la única información que se posee es la definición del problema. Como es de esperarse, existe una gran diferencia en eficiencia y rapidez entre los sistemas especializados para resolver un problema y un planificador independiente del dominio. Esto se debe principalmente a que en los primeros, se puede especificar una heurística previamente probada, que haya sido derivada de un razonamiento lógico humano o de algún trabajo de investigación, mientras que en el segundo, ésta se deriva automáticamente de la declaración del problema. Sin embargo, la flexibilidad ofrecida por los planificadores independientes del dominio al poder resolver una cantidad mucho mayor de problemas a diferencia de los sistemas especializados es la justificación de numerosos estudios para poder obtener heurísticas que mejoren la eficiencia y la rapidez de los mismos.

Capítulo 3

BÚSQUEDA

En este capítulo explicaremos con más detalle los espacios de búsqueda y los modelos de estados asociados a cada uno de ellos. Como vimos en el capítulo anterior, el modelo que ha sido utilizado desde hace muchos años para describir los problemas de planificación es STRIPS. En la planificación como búsqueda heurística, la idea principal consiste en traducir el modelo STRIPS, a un modelo de estados, el cual es resuelto mediante una búsqueda, utilizando una heurística que es derivada automáticamente de la definición del problema [18]. En este capítulo presentaremos dos modelos de estados, cada uno asociado a una dirección de búsqueda, ambos derivados del modelo de STRIPS, que han sido utilizados en la planificación como búsqueda heurística [7, 6].

3.1. Espacios de búsqueda

Como vimos anteriormente, para resolver un problema STRIPS mediante búsqueda heurística, es necesario plantear un modelo de estados. La definición de este modelo depende directamente de la dirección en la cual desea realizarse la búsqueda. En esta sección explicaremos ambas direcciones de búsqueda, *hacia adelante* y *en regresión*.

3.1.1. Búsqueda hacia adelante

La búsqueda hacia adelante es la forma de natural de abordar el problema STRIPS. Partiendo del estado inicial, se aplica la función de transición de estados hasta alcanzar uno de los objetivos. El modelo de estados asociado es derivado directamente de la definición del problema STRIPS, y se define como una tupla $S_P = \langle S, s_0, S_G, A(.), f, c \rangle$ donde:

- $s \in S$ son colecciones de átomos.
- $s_0 = I$ es el estado inicial.

- Los estados objetivo $s \in S_G$ son aquellos donde $G \subseteq s$.
- Las acciones $a \in A(s)$ son los operadores $op \in O$ tal que $Prec(op) \subseteq s$
- La función de transición f está definida como: $f(s,a) = s - Del(a) + Add(a)$ donde $a \in A(s)$
- Todos los costos de las acciones son iguales a 1.

un plan consiste en una secuencia de acciones $a_0, a_1, a_2, \ldots, a_n$ que a su vez definen una secuencia de estados $s_0, s_1 = f(a_0, s_0), \ldots, s_{n+1} = f(a_n, s_n)$ tal que cada acción a_i es aplicable para todo estado s_i y el último estado es objetivo, $s_{n+1} \in S_G$. El plan se considera óptimo si la suma de los costos es mínima.

3.1.2. Búsqueda en regresión

Como vimos en la sección 2.2.1, un problema de planificación puede ser resuelto realizando una búsqueda *hacia atrás*, a partir del objetivo hasta el estado inicial s_0. Esta idea ha sido estudiada desde hace muchos años [7, 30, 33] . Surge como solución a la baja tasa de generación de nodos, debida principalmente al cálculo de la heurística en cada nodo expandido. Bonet y Geffner [7] estudian a fondo esta idea y concluyen que el 80 % del tiempo de planificación *hacia adelante* está atribuido al cálculo de la heurística; pero en los experimentos que realizaron, ninguna dirección dominó a otra. Cabe destacar que en las heurísticas de bases de datos de patrones, como veremos en el próximo capítulo, el tiempo de construcción de las tablas PDB es significativo, incluso mucho mayor al tiempo para el cálculo de las heurísticas consideradas por Bonet y Geffner en su estudio, por esta razón es imprescindible realizar una *búsqueda en regresión*. Para comprender mejor el concepto de *búsqueda en regresión*, Bonet y Geffner [7] definen un modelo de estados, al cual llaman *espacio de regresión*, que tambien se deriva del problema STRIPS original. Recordemos que un problema STRIPS está definido por una tupla $\langle A, O, I, G \rangle$. El modelo de estados de regresión derivado es $R_P = \langle S, s_0, S_G, A(.), f, c \rangle$ donde:

- Los estados $s \in S$, al igual que el modelo de estados original, son conjuntos de átomos de A.
- El estado inicial s_0 es el conjunto objetivo G.
- Los estados objetivos $s \in S_G$ son todos los $s \subseteq I$.
- El conjunto de acciones $A(s)$ aplicables en un estado s son los operadores $op \in O$ tal que son *relevantes* y *consistentes*, es decir, $Add(op) \cap s \neq \varnothing$ y $Del(op) \cap s = \varnothing$.
- La función de transición se define como: $f(a,s) = s - Add(a) + Prec(a)$ para todo $a \in A(s)$.

- Los costos de todas las acciones son 1.

La solución del *espacio de regresión* es también una secuencia de acciones $a_0, a_1, \ldots, a_n$. Las soluciones del modelo utilizado en la *búsqueda hacia adelante* y la del *espacio de regresión* están relacionadas de forma obvia: Una es la inversa de la otra. [7]. En la *búsqueda en regresión*, los estados son considerados como conjuntos de sub-objetivos. Por ejemplo, la aplicación de una regresión en el conjunto objetivo G conlleva una situación en la cual la ejecución de dicha acción alcanza el objetivo. Mientras un estado $s = \{p, q, r\}$ en el espacio de progresión define un único estado del problema donde los átomos p, q y r son verdaderos, en el espacio de regresión define una *colección* de estados donde estos átomos son verdaderos.

Invariantes y regresión

La *búsqueda en regresión* muchas veces genera estados que no son alcanzables desde el estado inicial s_0. Por ejemplo, en el dominio Blocksworld, una regresión del estado

$$s = \{\texttt{(on c d)}, \texttt{(on a b)}\} \tag{3.1}$$

con la acción $moveToBlock(a, d, b)$, nos lleva al estado

$$s' = \{\texttt{(on c d)}, \texttt{(on a d)}, \texttt{(clear b)}, \texttt{(clear a)}\} \tag{3.2}$$

El estado s' es claramente inalcanzable dado que representa una situación en la cual dos bloques a y c están sobre el mismo bloque d. Para la *búsqueda hacia adelante* es fácil demostrar que esta situación es imposible de alcanzar desde el estado inicial s_0 dado el comportamiento de cada acción pero en la *búsqueda en regresión* estas situaciones son muy comunes y deben ser evitadas. La principal razón por la cual deben evitarse es que, si se continua la búsqueda, todo los estados generados a partir de s' serán inválidos y esto representa un serio problema, sobre todo cuando el buen uso de la memoria durante la búsqueda es vital. Una alternativa para detectar dichos estados inalcanzables es utilizar el concepto de invariantes.

Invariantes Un invariante se define como un conjunto de átomos que cumplen cierta propiedad. Generalmente, esta propiedad declara que a lo sumo uno de los átomos del conjunto es verdadero en todo estado alcanzable desde s_0. Existen muchos estudios para determinar de forma automática los invariantes de un problema [15, 12], la mayoría coincide con un análisis del dominio. Bonet y Geffner [7] definen un

tipo de invariante, los mutexes, los cuales consisten en pares de átomos mutuamente exclusivos. Determinar todos los mutexes, u otro tipo de invariante de un problema pueder ser tan complicado como la propia resolución del mismo [7]. Existen algoritmos aproximados, que detectan subconjuntos de mutexes. Más adelante discutiremos el utilizado en este proyecto.

Capítulo 4

HEURÍSTICAS

Como en todo problema de búsqueda, la resolución de la planificación como búsqueda heurística consiste en hallar un camino mínimo desde un nodo inicial hasta un nodo objetivo. La complejidad de este proceso es el tamaño del espacio de búsqueda, el cual es en muchos casos exponencial [16]. La idea principal de la búsqueda heurística es, en vez de probar todos los posibles caminos en el espacio de búsqueda, establecer un criterio para enfocarse en aquellos que se acercan más al objetivo. Para realizar este tipo de búsqueda es necesario obtener una función de evaluación que clasifique cada nodo del espacio de búsqueda de acuerdo a su distancia al objetivo. De esta necesidad surgen las funciones heurísticas, las cuales asignan a cada estado (nodo) del problema un valor que determina su prioridad de consideración en el proceso de búsqueda. Para garantizar la optimalidad de la solución es necesario que la función heurística sea admisible [30], es decir, que no sobre-estime la distancia al objetivo. Dado que es de interés para este proyecto encontrar soluciones óptimas, en este capítulo se estudian dos familias de heurísticas admisibles, que son aplicadas en planificación.

4.1. Heurísticas de relajación del problema

Las heurísticas basadas en la relajación del problema, como su nombre lo indica, consisten en simplificar o relajar el problema original con la finalidad de reducir el espacio de búsqueda. Una vez simplificado, el problema puede ser resuelto de forma óptima y el costo de la solución del mismo es utilizado como valor estimado de la solución del problema original [7]. La idea de derivar heurísticas a partir de relajaciones al problema original tiene una larga historia en la Inteligencia Artificial [7, 30, 31]. En las competencias de planificación *AIPS*, se han destacado numerosos planificadores que han utilizado heurísticas de relajación, entre ellos podemos citar *HSP* [6], *Graphplan* [4] y *FF* [21]. Un ejemplo de este tipo de heurísticas es la familia de heurísticas h^m las cuales están basadas en la siguiente relajación: el costo de alcanzar un conjunto C de átomos objetivos ($|C| \geq m$) es igual al de alcanzar el subconjunto de tamaño m más

costoso. Este enfoque ha sido investigado por [8, 7, 5]. Cabe detacar que la heurística h^2 fue utilizada por el planificador *HSP2.0* [6] en la competencia de planificación *AIPS2000* y su desempeño fue destacado. Como vimos anteriormente, la heurística h^m estima el costo de alcanzar un conjunto de átomos C desde un estado s considerando los subconjuntos $D \subseteq C$ de tamaño m. Llamemos $g^m(C, s)$ el estimado de h^m para el conjunto C desde el estado s.
$g^m(C, s)$ está definida por la siguiente ecuación:

$$g^m(C,s) = \begin{cases} 0 & \text{if } C \subseteq s, \text{ else} \\ min_{B \in R(C)}[1 + g^m(B,s)] & \text{if } |C| \leq m \\ max_{D \subset C, |D|=m} g^m(D,s) & \text{otherwise} \end{cases} \tag{4.1}$$

donde $B \in R(C)$ si B es el resultado de aplicar una regresión a partir del conjunto de átomos C utilizando alguna acción a.

4.2. Heurísticas de bases de datos de patrones

Una base de datos de patrones (PDB) es una función heurística, basada en memoria, obtenida al considerar sólo una parte del problema (el patrón) lo suficientemente pequeña como para ser resuelta para todo estado a través de una búsqueda exhaustiva. Los resultados de esta búsqueda son almacenados en una tabla en memoria y se define una función heurística al asociar a cada estado una entrada en la tabla (estado abstracto), para luego leer su valor correspondiente [16]. Estas heurísticas han sido aplicadas exitosamente en muchos problemas de búsqueda, como el 24-puzzle y el cubo de Rubik [25, 26], y en la planificación clásica STRIPS [10].

4.2.1. Bases de datos de patrones en planificación STRIPS

La aplicación de las bases de datos de patrones en la planificación como búsqueda heurística fue introducida por Edelkamp [10]. Aplicadas a la planificación STRIPS, los patrones pueden ser considerados como un subconjunto de átomos del problema, ignorando los demás átomos. El patrón, definido como un subconjunto de átomos A, define un problema de planificación simplificado, el cual llamaremos problema abstracto, donde los estados y las listas asociadas a los operadores son intersectadas con A [16]. Adicionalmente, el patrón define un mapa de abstracción $\varphi^A(s)$ que asocia a cada estado del problema original un estado en el problema abstracto. Esta relación es descrita como $\varphi^A(s) = A \cap s$. De la misma forma, φ^A define la heurística admisible h^A, la cual se calcula resolviendo el problema abstracto completamente,

a través de una búsqueda exhaustiva.

Por ejemplo, supongamos el siguiente estado del dominio Blocksworld: Se tiene una pila tres bloques, los cuales denotaremos A, B y C, con el bloque C sobre la mesa, y el bloque A en el tope de la pila. Este estado, en STRIPS, es descrito por el siguiente conjunto de átomos:

$$s = \{\texttt{(on-table A)}, \texttt{(on B A)}, \texttt{(on C B)}, \texttt{(clear C)}\}$$

Ahora bien, definamos un patrón $A = \{\texttt{(on-table A)}, \texttt{(on A B)}, \texttt{(on A, C)}\}$. El estado abstracto asociado a s es: $\varphi^A(s) = \{\texttt{(on-table A)}\}$, ya que es el único átomo presente en el patrón.

En el caso de los operadores, el operador $MoveToTable(A, B)$, definido como:

$$moveToTable(A,B) = \begin{cases} precondition = \{\texttt{(clear A)}, \texttt{(on A B)}\} \\ addList = \{\texttt{(clear B)}, \texttt{(on-table A)}\} \\ delList = \{\texttt{(on A B)}\} \end{cases}$$

luego de ser proyectado sobre el patrón A, se obtiene:

$$moveToTable(A,B) = \begin{cases} precondition = \{\texttt{(on A B)}\} \\ addList = \{\texttt{(on-table A)}\} \\ delList = \{\texttt{(on A B)}\} \end{cases}$$

Como se explicó anteriormente, para calcular la heurística h^A debemos proyectar el problema sobre el patrón A y resolverlo mediante una búsqueda exhaustiva. En STRIPS, un problema $P = \langle S, O, I, G\rangle$ proyectado sobre un patrón A nos lleva a obtener el problema $P^A = \langle S \cap A,\ O^A,\ I \cap A,\ G \cap A\rangle$, donde O^A contiene todos los operadores $o \in O$ proyectados sobre A. El valor de la solución óptima en el problema abstracto P^A es una cota inferior del valor de la solución para el problema original P, como puede observarse en la ecuación 4.2.

$$h^A(s) \leq h^*(s) \tag{4.2}$$

Cualquier sub-conjunto A' de A define una abstracción del problema P^A a un problema (más) abstracto definido por A', de la misma forma como fue aplicado para el problema original P. Ésto implica a su vez que la solución óptima para el problema $P^{A'}$ es menor o igual a la solución en el

problema P^A [17]. Lo cual nos lleva a plantear la siguiente relación:

$$h^{A'}(s) \leq h^A(s) \text{ para todo } s, \text{ cuando } A' \subseteq A. \tag{4.3}$$

De las ecuaciones anteriores, podemos plantear la siguiente relación:

$$\max(h^A(s), h^B(s)) \leq h^{A \cup B}(s) \text{ para todo } s \tag{4.4}$$

Sin embargo, Bonet, Geffner y Haslum [17] plantean que bajo ciertas condiciones, la relación anterior puede ser ampliada a:

$$\max(h^A(s), h^B(s)) \leq h^A(s) + h^B(s) \leq h^{A \cup B}(s) \text{ para todo } s \tag{4.5}$$

siempre y cuando los patrones A y B sean aditivos. La definición de aditividad de patrones es introducida por [17] y explica que dos patrones son aditivos si ninguna acción del problema agrega átomos de ambos conjuntos.

La ecuación 4.3 indica que la calidad de la heurística puede mejorar a medida que se consideran patrones de mayor tamaño. Sin embargo, la cantidad de memoria que debemos utilizar para calcular y almacenar h^A es exponencial en el número de átomos del conjunto A [16]. Este hecho nos plantea el siguiente problema:
Si bien la aditividad de patrones nos permite obtener heurísticas sin incrementar el uso de la memoria, dado que la cantidad de memoria utilizada para calcular $\max(h^A, h^B)$ es la misma para $(h^A + h^B)$, la ecuación 4.5 nos indica que al calcular $h^{A \cup B}$ podemos obtener una heurística aún mejor. Sin embargo, la cantidad de memoria necesaria es mucho mayor a la anterior. Seleccionar el conjunto de patrones que nos permita obtener la mejor heurística, dentro de los límites de memoria permitidos, es un problema complicado. Este problema es discutido más adelante, en la sección 4.2.4.

4.2.2. Representación en variables multi-valuadas

Edelkamp [10] propone, para optimizar el uso de la memoria y la consulta de la PDB, utilizar una nueva representación en variables multi-valuadas, que están definidas implícitamente en el problema. Tales variables corresponden a un tipo de invariante particular: La propiedad de que al lo sumo un

átomo del conjunto pertenece a todo estado alcanzable del problema. La razón por la cual basar los patrones en variables multi-valuadas hace más efectivo el uso de la memoria es la siguiente: *Un patrón general de n átomos define* 2^n *estados abstractos y el tamaño de la tabla debe ser* 2^n*, mientras que, un patrón basado en una variable de n valores (n átomos mutuamente exclusivos), define n estados abstractos y el tamaño de la tabla, por consiguiente, será de simplemente n entradas* [16].
La representación SAS, introducida por Bäckström y Nebel [2], fue utilizada por Helmert [19] para definir un modelo basado en variables multi-valuadas, como una alternativa al modelo STRIPS.
Un problema de planificación en SAS es una tupla $P = \langle V, O, s_0, s_* \rangle$ tal que:

- $V = \{V_1, V_2, \ldots, V_n\}$ es un conjunto de variables de estado. Un estado se define como una asignación de valores a cada variable.
- O es un conjunto de operadores, donde un operador $op \in O$ es un par $\langle pre, ef \rangle$ de asignaciones parciales de variables llamados precondición y efecto respectivamente.
- s_0 es un estado llamado *estado inicial* y
- s_* es una asignación parcial de variables, llamado objetivo.

Un operador $op = \langle pre, ef \rangle$ es aplicable en un estado s si toda variable $V_i = v_i$ en la asignación parcial pre, tiene el mismo valor v_i en el estado s. La aplicación del operador, modifica el valor de V_i si existe una asignación para V_i en ef. La nueva representación de variables multi-valuadas servirá para explicar el enfoque de las bases de datos de patrones que es dado en este trabajo a la planificación.

4.2.3. PDB con restricciones

Como fue explicado anteriormente, un patrón P consiste en un conjunto de átomos del problema. Al generar una abstracción, proyectando el problema original sobre el patrón, se están ignorando los átomos que no están en P y al realizar la búsqueda exhaustiva para calcular la PDB, es usual que se subestime la distancia real para un estado dado. La principal razón es que al omitir un conjunto de átomos del problema y realizar la búsqueda, se pueden violar aquellos invariantes que involucren estos átomos. Para resolver este problema, Bonet, Geffner y Haslum [17] introducen las PDB con restricciones, las cuales se definen de la siguiente forma:
Sea $C = \{C_1, C_2, \ldots, C_n\}$ una colección de invariantes, sea P^A un problema abstracto con respecto a un patrón A y $R(P^A)$ el espacio de búsqueda en regresión asociado al problema P^A. Se define el espacio de búsqueda en regresión restringido $R_C(P^A)$ como un sub-espacio de $R(P^A)$ que contiene sólo los estados

que satisfacen los invariantes en C y las aristas (s, a, s') exceptuando aquellas que $s' \cup pre(a)$ viole algún invariante en C. En pocas palabras, la abstracción restringida excluye los estados que se saben son inalcanzables y aquellas transiciones que no son aplicables en el problema original.

Teorema 4.2.1 *El costo óptimo en el espacio de búsqueda restringido es una heurística admisible para el problema original y es al menos tan fuerte como la heurística obtenida de la abstracción sin restricciones. [17]*

$$h^A(s) \leq h^A_C(s) \leq h^*(s) \tag{4.6}$$

Prueba El planteamiento de este teorema y su demostración se basa en la correspondencia que existe entre una solución en el espacio de regresión $R(P)$ y una secuencia de acciones aplicables desde el estado inicial. Todo camino en $R(P)$ tiene una proyección en $R(P^A)$, así como la solución óptima del problema, la cual satisface los invariantes en C.

Ejemplo: Blocksworld Consideremos el siguiente problema: Se tienen 4 bloques (A, B, C, D) donde A y C se encuentran directamente sobre la mesa, el bloque B sobre A y el bloque D sobre C. El objetivo es conformar una torre $A \rightarrow B \rightarrow C \rightarrow D$, donde D sea la base y A se encuentre en el tope. El estado inicial y la situación final se describen como:

$$s_0 = \{\texttt{(on-table A)}, \texttt{(on B A)}, \texttt{(on-table C)}, \texttt{(on D C)}\}$$
$$G = \{\texttt{(on-table D)}, \texttt{(on C D)}, \texttt{(on B C)}, \texttt{(on A B)}\}$$

Para ejemplificar las bases de datos con restricciones, definamos un conjunto C de invariantes como:

$$C = \begin{cases} \{\texttt{(on X Y)(on Y X)}\} \\ \{\texttt{(on X Z)(on Y Z)}\} \quad \forall\ X,\ Y\ =\ A, B, D, C \\ \{\texttt{(on Z X)(on Z Y)}\} \end{cases} \tag{4.7}$$

Los elementos de C son pares de átomos exclusivos. En la práctica, estos pares se identifican utilizando la heurística h^2, dado que el valor de la ésta, para todo conjunto $\{p, q\} \in C$ es infinito. Las variables del problema son similares a las definidas en 5.4, pero en este caso para 4 bloques. Ahora bien, consideremos

un patrón P conformado por las siguientes variables:

$$pos(C) = \{\texttt{(on-table C)}, \texttt{(on C A)}, \texttt{(on C B)}, \texttt{(on C D)}\}$$

$$pos(D) = \{\texttt{(on-table D)}, \texttt{(on D A)}, \texttt{(on D B)}, \texttt{(on D C)}\}$$

Las tablas para el patrón $P = pos(C) \cup pos(D)$, utilizando bases de datos con restricciones y sin restricciones se muestran en la tabla 4.1.

$pos(C)$	$pos(D)$	Valor en $R(P)$	Valor en $R_C(P)$
(on-table C)	(on-table D)	1	1
(on-table C)	(on D C)	0	0
(on-table C)	(on D B)	1	1
(on-table C)	(on D A)	1	1
(on C D)	(on-table D)	2	2
(on C D)	(on D C)	**1**	∞
(on C D)	(on D B)	2	2
(on C D)	(on D A)	2	2
(on C B)	(on-table D)	2	2
(on C B)	(on D C)	**1**	**3**
(on C B)	(on D B)	**2**	∞
(on C B)	(on D A)	2	2
(on C A)	(on-table D)	2	2
(on C A)	(on D C)	**1**	**3**
(on C A)	(on D B)	2	2
(on C A)	(on D A)	**2**	∞

Tabla 4.1: Comparación de PDBS con y sin restricciones.

Como podemos apreciar, ambas tablas difieren en 5 entradas. Lo cual implica que para todos los estados s, donde las variables $pos(C)$ y $pos(D)$ tengan valores iguales a alguna de estas entradas, se cumple la desigualdad estricta $h^P(s) < h^P_C(s)$.

4.2.4. Selección de Patrones

En la sección 4.2.1 se explicó que la ecuación 4.3 indica que la calidad de la heurística puede mejorar a medida que se consideran patrones de mayor tamaño. Sin embargo, el principal problema de las heurísticas PDB es que el tamaño en memoria y la cantidad de tiempo necesaria para la construcción de las tablas limitan el tamaño del patrón. Por esta razón la calidad de la heurística depende de forma crucial en la selección de los patrones. En la planificación independiente del dominio, donde el patrón apropiado varía entre cada problema, la selección debe ser automática, lo cual puede llegar a ser bastante problemático.

Veamos el siguiente ejemplo:

- Supongamos que tenemos un conjunto de 6 variables $(V_1, V_2, \ldots, V_6)$
- Como vimos en la sección 4.2.2, para optimizar el uso de la memoria y la consulta de la PDB, debemos basar los patrones en las variables. Por ello, definimos un patrón por cada variable: $(P_1 = \{V_1\}, P_2 = \{V_2\}, \ldots, P_6 = \{V_6\})$
- Construimos h^{P_i} para cada patrón.
- En este momento se podría definir la heurística $h^{PDB}(s) = max(h^{P_1}(s), h^{P_2}(s), \ldots, h^{P_6}(s))$. Pero no estaríamos aprovechando la propiedad de aditividad de patrones. Supongamos que obtenemos: $P = \{\{P_1, P_2, P_3\}, \{P_4, P_5\}, \{P_6\}\}$, donde cada conjunto de P es un conjunto de patrones aditivos entre si.
- Aprovechando la aditividad, definimos la nueva heurística, mejor que la anterior, como: $h_{PDB}(s) = max(h^{P_1}(s) + h^{P_2}(s) + h^{P_3}(s),\ h^{P_4}(s) + h^{P_5}(s),\ h^{P_6}(s))$.

En el ejemplo anterior, si bien estamos aprovechando la aditividad para obtener mejores heurísticas, no estamos utilizando la ecuación 4.3, que nos indica que incrementando el tamaño del patrón podemos obtener heurísticas aún mejores. Veamos el siguiente ejemplo:

- Supongamos que tenemos un conjunto aditivo de 4 patrones: $S = \{A, B, C, D\}$
- Construimos h^x para $x = A, B, C, D$.
- En este momento se podría definir la heurística $h^{PDB}(s) = max(h^A(s), \ldots, h^D(s))$. Pero aprovechando la aditividad, definimos la nueva heurística, mejor que la anterior, como: $h_{PDB}(s) = h^A(s) + h^B(s) + h^C(s) + h^D(s)$.
- Supongamos que el tamaño de cada patrón es n. Es decir, el tamaño de la tabla asociada a cada heurística h^x es igual a n. Si el límite de tamaño establecido para cada patrón es n^3, podemos agrupar 2 y hasta 3 patrones para conformar patrones de mayor tamaño, lo cual nos permitiría obtener una heurística mejor, según la ecuación 4.3.
- Esto nos lleva al siguiente problema: Elegir cual de las siguientes selecciones nos permite obtener

la mejor heurística:

$$h_{PDB}(s) = h^{A \cup B}(s) + h^{C \cup D}(s)$$
$$h_{PDB}(s) = h^{A \cup C}(s) + h^{B \cup D}(s)$$
$$h_{PDB}(s) = h^{A \cup D}(s) + h^{B \cup C}(s)$$
$$h_{PDB}(s) = h^{A \cup B \cup C}(s) + h^{D}(s)$$
$$h_{PDB}(s) = h^{A \cup B \cup D}(s) + h^{C}(s)$$
$$h_{PDB}(s) = h^{A \cup D \cup C}(s) + h^{B}(s)$$
$$h_{PDB}(s) = h^{D \cup B \cup C}(s) + h^{A}(s)$$

Como podemos observar en los ejemplos anteriores, para aprovechar la aditividad y el incremento del tamaño del patrón, considerando los límites de memoria y tiempo, con la finalidad de obtener mejores heurísticas, nos debemos enfrentar a un problema de selección de patrones. Este problema había sido estudiado de forma parcial por [10, 22]. Bonet, Geffner y Haslum [17] lo abordan con profundidad y plantean una estrategia de selección de patrones, llamada *Selección Iterativa Incremental*, la cual explicaremos más adelante en la sección 4.2.4.

Esquema general para la selección de patrones

En el método propuesto por Bonet, Geffner y Haslum [17] existe, de forma implícita, un esquema general para los métodos de selección de patrones. El mismo consiste en, partiendo de conjuntos aditivos de variables, crear un patrón por cada variable e ir seleccionando pares de patrones para unirlos y conformar nuevos patrones. Esto permite que la heurística final maximice sobre las sumas de cada conjunto aditivo, manteniendo la admisibilidad. El proceso se describe a continuación:

1. Se determinan conjuntos aditivos de variables.
2. Se construye una PDB para cada variable. Es decir, se crea un patrón por cada variable.
3. Se aplica el siguiente proceso para cada conjunto aditivo:
   - *a*) Se analizan los conflictos entre patrones.
   - *b*) Se unen aquellos patrones que tengan mayor conflicto.
   - *c*) El proceso se repite hasta que no hayan más conflictos o hasta que no se puedan unir patrones por restricciones de memoria.

4. La heurística final consiste en la maximización de la suma de cada conjunto aditivo.

Como podemos observar, este esquema general nos permite:

- Aprovechar la aditividad, determinando conjuntos maximales de patrones aditivos, y
- Incrementar el tamaño de los patrones, analizando conflictos para seleccionar cuales patrones unir, para obtener mejores heurísticas.

Como se podrá observar más adelante, este esquema general de selección de patrones es utilizado por los métodos discutidos en este proyecto. El punto en que difieren es el análisis de conflictos, que determina los patrones a seleccionar.
Una condición importante para considerar la unión de patrones es el tamaño de la nueva PDB. Dados dos patrones A y B y el número de entradas de las tablas $|A|$ y $|B|$ respectivamente, el número de entradas de la nueva PDB asociada al patrón $A \cup B$ es igual a $|A|x|B|$, lo cual representa un incremento en el uso de memoria. Para manejar este inconveniente, se implementó un manejador de memoria, al cual se le establece un límite L. Al analizar los conflictos, se ignoran aquellos que violen la cantidad de memoría disponible. Cabe señalar que todas las PDB son restringidas y se utilizan los mutex como colección de invariantes.

Selección incremental de patrones

Este método, junto al esquema general de selección de patrones es propuesto por Bonet, Geffner y Haslum [17]. Consiste en análizar posibles conflictos entre patrones a través de las soluciones de los problemas abstractos y unir aquellos patrones que aparenten ser más interdependientes.

Recordemos que una PDB se construye realizando una búsqueda exhaustiva en regresión. Durante este proceso podemos mantener un apuntador en cada estado abstracto para indicar cuales acciones fueron aplicadas para llegar a ese estado. Esto nos permitirá extraer rápidamente el plan abstracto, que representa una solución para el problema proyectado. Para un patrón A que consista en un conjunto de variables, los posibles conflictos pueden ser detectados extrayendo el plan abstracto y simulando este plan desde el estado inicial del problema original utilizando las acciones originales. Cuando la precondición de una acción del plan abstracto se encuentra inconsistente con el valor de una variable V, que no pertenece al patrón, se detecta un conflicto entre el patrón A y el patrón al cual pertenece la variable V. Esta inconsistencia puede ser por las siguientes razones:

- El valor de V de la precondición contradice el valor del estado s al cual se desea aplicar la acción. Esto, traducido en el ambiente STRIPS, representa que la acción a no es aplicable en s. ($pre(a) \nsubseteq s$)
- La precondición y el valor de V unidos, violan algún invariante.

El proceso completo, aplicado junto al esquema general, se describe de la siguiente forma:
Para cada conjunto aditivo de variables $\{V_1, \ldots, V_n\}$, se crea una colección de patrones $C = \{P_1, \ldots, P_n\}$, donde cada patrón contiene una variable. Se construyen las PDB para cada patrón, se extraen los planes abstractos y se analizan los conflictos. Los conflictos con patrones que no estén en C son ignorados. Cada conflicto se pondera utilizando el valor de h^1 del conjunto de átomos inconsistente. Se toma el conflicto con mayor peso y se realiza el proceso de unión de patrones.

Gap at init

Esta estrategia para la selección de patrones es propuesta e implementada en este proyecto. La idea principal es determinar si $h^{A\cup B}(s)$ es superior a $h^A(s) + h^B(s)$ (véase ecuación 4.5) sin necesidad de construir la PDB para el patrón $A \cup B$. Para evitar tener que construir la PDB se utiliza el algoritmo IDA* para resolver el problema proyectado $P^{A\cup B}$, en el espacio de regresión $R_C(P^{A\cup B})$ y se utiliza $h(s) = max(h^2(s), h^A(s)+h^B(s))$ como heurística. La longitud de la solución encontrada corresponde al valor de $h^{A\cup B}(G)$. Los otros dos valores (h^A y h^B) son obtenidos de las tablas ya existentes de estos dos patrones. Luego ponderamos el conflicto entre A y B con el valor de la diferencia $h^{A\cup B}(G) - h^A(G) - h^B(G)$. Este proceso de realiza para cada par de patrones. Al final se selecciona el conflicto con mayor peso y se realiza la unión. Una variante de esta estrategia es considerar trios de patrones (A, B, C), utilizando la heurística $h(s) = max(h^A(s)+h^B(s)+h^C(s), h^2(s))$. Cabe destacar que la resolución del problema $P^{A\cup B}$ es muy diferente al cálculo de $h^{A\cup B}$ ya que en el primero sólo se obtiene la longitud del plan, es decir $h^{A\cup B}(G)$, mientras que el segundo se corresponde a hallar el camino mínimo entre el estado inicial s_0 y cada estado s. En la experimentación se observó que el algoritmo IDA* converge rápidamente al utilizar la heurística antes mencionada.El método propuesto para esta estrategia está descrito en el algoritmo 1.

Gap set

Esta estrategia, al igual que la anterior, es propuesta e implementada en este proyecto. Principalmente surge como una mejora al *Gap at init*. Consiste en, dados dos patrones A y B, y un conjunto de estados S cercanos al objetivo, ponderar el conflicto entre A y B como la sumatoria de los diferenciales de $h^{A\cup B}(s) - h^A(s) - h^B(s)$, para todo estado $s \in S$. El número de estados en S es un parámetro de este algoritmo y en la experimentación se utilizarán diferentes valores. El algoritmo utilizado para hallar el

Algoritmo 1: Gap at init

$V = \{V_1, \ldots, V_n\}$ es el conjunto de variables aditivas
P es un conjunto de patrones, inicialmente vacio
forall $V_i \in V$ **do**
    Definimos un patrón P_i, con la variable V_i
    $P = P \cup \{P_i\}$
    Se construye la tabla PDB_i para el patrón P_i
    Se crea la heurística h^{P_i}
end
while *Exista memoria disponible* **do**
    forall $P_i \in P$ **do**
        forall $P_j \in P$ *y* $i \neq j$ **do**
            if $|PDB_i| \times |PDB_j| >$ *memoria disponible* **then**
                continuar
            end
            $P_{ij} = P_i \cup P_j$
            Proyectamos el problema original sobre P_{ij}
            Hallamos el costo óptimo c de resolver el problema proyectado, utilizando la heurística $h(s) = max(h^{P_i}(s) + h^{P_j}(s), h^2(s)$
            Poderamos el conflicto entre P_i y P_j con el valor $c - h^{P_i} - h^{P_j}$
        end
    end
    Seleccionamos el conflicto (P_i, P_j) con mayor peso
    Definimos un nuevo patrón $P_{ij} = P_i \cup P_j$
    $P = P \setminus \{P_i, P_j\}$
    $P = P \cup \{P_{ij}\}$
    Construimos la tabla PDB_{ij} para el nuevo patrón
    Creamos la heurística h_{ij}
end

valor de $h^{A\cup B}(s)$ es IDA*, empezando la búsqueda en el estado s, en el espacio de regresión $R_C(P^{A\cup B})$, utilizando $h(s) = max(h^A(s) + h^B(s), h^2(s))$ como heurística, este procedimiento se aplica para todo $s \in S$.

Para hallar el conjunto de estados S se utilizó A*, en el espacio de regresión, con la heurística h^2 para evitar generar estados inalcanzables. Se pasa como parámetro el número de estados m que se desea obtener y el algoritmo termina cuando $|S| \geq m$. El método propuesto para esta estrategia está descrito en el algoritmo 2.

Detalles de implementación En la búsqueda en regresión, como vimos en la sección 3.1.2, los estados son considerados como conjuntos de sub-objetivos. Ésto, en la representación SAS, implica que los estados pueden ser asignaciones parciales (no todas las variables tienen valor). Para el cálculo de la PDB, como se podrá ver más adelante, el valor de $h^A(s)$ para una asignación parcial es el mínimo de todas las posibles completaciones. Dada esta afirmación, se plantea la necesidad de que todo estado $s \in S$ sea una asignación completa de variables, de lo contrario, el costo mínimo hallado por el algoritmo de búsqueda no representará el valor de $h^{A\cup B}(s)$. La manera como este problema es resuelto es completando todos los estados en S mediante un algoritmo recursivo, el cual establece valores para las variables sin asignación, hasta obtener una asignación completa. Se utiliza la heurística h^2 para evitar obtener estados inalcanzables. Otro detalle de implementación es la necesidad de optimizar el algoritmo evitando recalcular conflictos de forma innecesaria. Luego de cada iteración, dos patrones son unidos y constituidos en un nuevo patrón. Los pesos de los conflictos entre los demás patrones no deben ser recalculados, ya que la unión de los anteriores no afecta a éstos. Los únicos conflictos que deben ser ponderados son aquellos que involucran al nuevo patrón creado. De esto podemos concluir que el número de llamadas al algoritmo IDA* es: Siendo V el conjunto de variables aditivas y S el conjunto de estados cercanos al objetivo.

- $(|V| \times (|V| - 1) \times |S|)/2$ en la primera iteración.
- $((|V| - i) \times |S|)/2$ en la iteración i.

Discusión Esta estrategia está enfocada en determinar si $h^{A\cup B}$ es superior a $(h^A(s) + h^B(s))$ de la forma más simple: tomando una muestra de estados S y obtener los valores de $h^{A\cup B}(s)$ para todo $s \in S$. Si bien, para una tabla de 2.000.000 de entradas una muestra de 250 estados representa menos del 1 %, en las primeras iteraciones, cuando las tablas tienen menor número de entradas, representaría un alto porcentaje. En los experimentos se observó que en las primeras iteraciones, son numerosos los conflictos ponderados con valores mayores a 0, pero a medida que avanza el algoritmo, y las tablas aumentan de

tamaño, es común que la mayoría de los conflictos tengan peso igual a cero. Un aspecto positivo es que las primeras iteraciones pueden ser consideradas determinantes para la calidad de la heurística, dado que son las decisiones con más opciones, a diferencia de las últimas iteraciones, en las cuales el número de patrones es mucho menor y el número de opciones para realizar las uniones son menores dado el límite de memoria.

Algoritmo 2: Gap set

$V = \{V_1, \ldots, V_n\}$ es el conjunto de variables aditivas
P es un conjunto de patrones, inicialmente vacio
S es un conjunto de asignaciones completas, que definen estados cercanos al objetivo
forall $V_i \in V$ **do**
    Definimos un patrón P_i, con la variable V_i
    $P = P \cup \{P_i\}$
    Se construye la tabla PDB_i para el patrón P_i
    Se crea la heurística h^{P_i}
end
while *Exista memoria disponible* **do**
    forall $P_i \in P$ **do**
        forall $P_j \in P$ *y* $i \neq j$ **do**
            if $|PDB_i| \times |PDB_j| >$ *memoria disponible* **then**
                continuar
            end
            $P_{ij} = P_i \cup P_j$
            Proyectamos el problema original sobre P_{ij}
            acum = 0
            forall $s \in S$ **do**
                Hallamos el costo óptimo c de resolver el problema proyectado, a partir del estado s, utilizando la heurística $h(s) = max(h^{P_i}(s) + h^{P_j}(s), h^2(s)$
                acum = acum + $(c - h^{P_i}(s) - h^{P_j}(s))$
            end
            Poderamos el conflicto entre P_i y P_j con el valor de acum
        end
    end
    Seleccionamos el conflicto (P_i, P_j) con mayor peso
    Definimos un nuevo patrón $P_{ij} = P_i \cup P_j$
    $P = P \setminus \{P_i, P_j\}$
    $P = P \cup \{P_{ij}\}$
    Construimos la tabla PDB_{ij} para el nuevo patrón
    Creamos la heurística h_{ij}
end

Capítulo 5

IMPLEMENTACIÓN

Este capítulo presenta una descripción general de la implementación del planificador. Para ello, el capítulo se dividió en dos secciones. En la primera se presentarán los detalles de implementación del planificador. Luego se describen algunos detalles deimplementación de las heurísticas, así como de los métodos de selección de patrones.

5.1. Planificador

Para desarrollar el proyecto e implementar las diferentes heuírsticas, se implementó un planificador similar a *HSP2.0* [6]. Para entender los detalles de implementación, el sistema fue dividido en un conjunto de sub-sistemas, donde cado uno recibe una información de entrada, la procesa, y genera una salida que será utilizada por otro sub-sistema:

1. Parser
2. Compilador STRIPS
3. Generador del problema de búsqueda
4. Resolución de la búsqueda

5.1.1. Parser

Un planificador, como todo solucionador general de problemas, debe proveer un lenguaje para describir y modelar el problema a un alto nivel. Es por ello que en la implementación, se utiliza el lenguaje PDDL (Planning Domain Definition Language) para obtener la definición del problema. Por esta razón, el primer elemento del proceso de planificación es el parser. Su principal objetivo es procesar el archivo

PDDL, en texto plano, que contiene la descripción completa del problema. Para entender el proceso, primero de presentará la descripción de PDDL, luego las herramientas que se utilizaron para elaborar el parser y por último se discutirá la salida de este sub-sistema.

PDDL

PDDL (Planning Domain Definition Language) es un lenguaje para describir problemas de planificación. Fue desarrollado en 1998 por Drew McDermont [28] para llevar a cabo la primera competencia de planificación *AIPS98*. PDDL utiliza el modelo ADL (Action Definition Language) [32] para definir los problemas. Un problema en ADL, es una tupla $P = \langle A, I, G, O \rangle$ donde:

- A es un conjunto de átomos.
- $I \subseteq A$ es el estado inicial.
- G es una fórmula proposicional que denota los estados objetivo.
- Los operadores $o \in O$ son de la forma (P, E) donde P es una fórmula proposicional que denota la precondición y E es el efecto de o.

En este proyecto no fueron utilizadas todas las características del módelo ADL. Las explicadas a continuación son soportadas por el planificador:

- **Cuantificadores en el estado objetivo:** Dado que el estado objetivo es una fórmula proposicional que describe los átomos presentes en el conjunto objetivo, se puede utilizar un cuantificador universal en la especificación. Esto facilita el trabajo cuando el conjunto objetivo es muy numeroso.
- **Sistemas de tipos en las variables:** Se puede utilizar una jerarquía de tipos para describir las acciones.

Volviendo a PDDL, un problema en este lenguaje consta de dos partes: el *dominio*, que define un conjunto de predicados, constantes y esquemas de acciones, y el *problema*, que describe los objetos, la situación inicial y los objetivos. A continuación, se presentan algunas definiciones:

- **Predicado:** un predicado puede considerarse un prototipo de átomo. En el dominio de *Bloksworld*, donde hay tres bloques A, B y C, los predicados serían: `(on-table ?x)`, `(on ?x ?y)`, `(clear ?x)`, donde

?x, ?y son variables. Mientras que los átomos, luego de una instanciación de los predicados, serían:

$$\{\texttt{(on-table A)}, \texttt{(on-table B)}, \texttt{(on-table C)},$$
$$\texttt{(on A B)}, \texttt{(on A C)}, \texttt{(on B A)}, \texttt{(on B C)}, \texttt{(on C B)},$$
$$\texttt{(on C A)}, \texttt{(clear A)}, \texttt{(clear B)}, \texttt{(clear C)}\}$$

- **Esquema de acciones:** Una acción descrita en PDDL consta de una lista de variables, una precondición y un efecto. Las dos últimas definidas como una fórmula proposicional. En la versión de PDDL que es soportada por el planificador, la precondición sólo puede ser una conjunción de predicados. De la misma forma, el efecto sólo puede ser una conjunción de predicados o negaciones de predicados.
- **Constantes:** Son objetos que están presentes en todos los problemas del dominio.
- **Situación Inicial:** La situación inicial es descrita como un conjunto de átomos.
- **Conjunto de estados objetivo:** Son descritos como una formula proposicional de átomos.
- **Flags:** Indican la utilización de algunas opciones al planificador. Los flags soportados son: **equality**, para indicar que se pueden repetir objetos en la instanciación de predicados y **typing** para indicar la utilización de un sistema de tipos.

Ejemplo: Gripper con Tipos
Dominio:

```
(define (domain gripper-typed)
(:requirements :typing)
(:types room ball gripper)
(:constants left right - gripper)
(:predicates (at-robby ?r - room)
             (at ?b - ball ?r - room)
             (free ?g - gripper)
             (carry ?o - ball ?g - gripper))
(:action move
         :parameters (?from ?to - room)
         :precondition (at-robby ?from)
         :effect (and (at-robby ?to) (not (at-robby ?from))))
```

⋮

Problema:

```
(define (problem strips-gripper-x-1)
   (:domain gripper)
   (:objects rooma roomb - room ball4 ball3 ball2 ball1 -ball left right - gripper)
   (:init (at-robby rooma)
          (free left)
          (free right)
          (at ball4 rooma)
          (at ball3 rooma)
          (at ball2 rooma)
          (at ball1 rooma)

   (:goal (and (at ball4 roomb)
               (at ball3 roomb)
               (at ball2 roomb)
               (at ball1 roomb))))
```

Implementación

Para la implementación, se utilizó como guía el parser implementado por *HSP2.0* [6]. En esta ocasión fueron utilizadas las herramientas *BISON* y *FLEX* [1] para elaborar el analizador léxico y sintáctico respectivamente. La gramática de los componentes de PDDL soportados por el planificador está descrita en [14].

Salida

La salida generada es una representación lógica del dominio y el problema a resolver.

5.1.2. Compilador STRIPS

El componente del sistema que hemos llamado *Compilador STRIPS* realiza los siguientes procesos:

- Instanciación de las acciones
- Procesamiento de las acciones
- Obtención de átomos alcanzables

Instanciación de las acciones

Como vimos anteriormente, las acciones en PDDL, recibidas por el planificador, se encuentran parametrizadas por variables. Para describir el problema general en STRIPS, es necesario instanciar cada una de estas acciones con los objetos y constantes presentes en el problema. Veamos un ejemplo:

Blocksworld

```
(define (domain blocks_world)
  (:requirements :strips)
  (:predicates (on-table ?x)
               (on ?x ?y)
               (clear ?x))

  (:action MoveToTable
      :parameters (?x ?y)
      :precondition (and (clear ?x) (on ?x ?y))
      :effect (and (clear ?y)
   (on-table ?x)
   (not (on ?x ?y))))

   ⋮
```

```
(define (problem ejemplo)
  (:domain blocks_world)
  (:objects A B C)
  (:init (on-table A) (clear C) (on C A)
 (on-table B) (clear B))
  (:goal (and (on A B) (clear A) (on-table B) (on-table C)))
  )
```

Instanciando la acción `(MoveToTable ?x ?y)`, para los objetos A, B y C, obtenemos:

```
+ (MoveToTable A B)
precondition: (and (clear A) (on A B))
effect: (and (clear B) (on-table A) (not (on A B)))

+ (MoveToTable A C)
precondition: (and (clear A) (on A C))
```

```
effect: (and (clear C) (on-table A) (not (on A C)))

+ (MoveToTable B A)
precondition: (and (clear B) (on B A))
effect: (and (clear A) (on-table B) (not (on B A)))

+ (MoveToTable B C)
precondition: (and (clear B) (on B C))
effect: (and (clear C) (on-table B) (not (on B C)))

+ (MoveToTable C A)
precondition: (and (clear C) (on C A))
effect: (and (clear A) (on-table C) (not (on C A)))

+ (MoveToTable C B)
precondition: (and (clear C) (on C B))
effect: (and (clear B) (on-table C) (not (on C B)))
```

Este proceso se realiza con un simple algoritmo recursivo que va asignando valores a cada variable. Cabe destacar que si el dominio no posee el flag **equality**, no se asignarán valores iguales a las variables. Cuando el dominio posee un sistema de tipos, éstos deben tomarse en cuenta a la hora de realizar las asignaciones.

5.1.3. Procesamiento de las acciones

Como vimos anteriormente, una acción STRIPS posee una precondición, una lista de átomos a agregar y otra a eliminar. Las acciones recibidas por el planificador se encuentran en ADL, y los efectos de las mismas son descritos como una fórmula proposicional, que posee únicamente conjunciones y negaciones. Para traducirlas a STRIPS, el proceso es muy simple: Los átomos que se encuentren negados en el efecto se agregarán a la lista de eliminación, los restantes a la lista de átomos a agregar.

Ejemplo:

```
+ (MoveToTable C B)
precondition: (and (clear C) (on C B))
effect: (and (clear B) (on-table C) (not (on C B)))
```

Se traduce a:

$$moveToTable(C,B) = \begin{cases} precondition = \{\texttt{(clear C)}, \texttt{(on C B)}\} \\ addList = \{\texttt{(clear B)}, \texttt{(on-table C)}\} \\ delList = \{\texttt{(on C B)}\} \end{cases}$$

Obtención de átomos alcanzables

La finalidad de este proceso es obtener los átomos alcanzables del problema. Un átomo alcanzable es aquel que pertenece a algún estado del problema. Para ello, se aplican las acciones, ya instanciadas, a partir del estado inicial sin tomar en cuenta la lista de eliminación, hasta obtener todos los átomos del problema.

Algoritmo 3: Obtención de átomos alcanzables

```
S ← I
listo ← FALSE
while not listo do
    x ←| S |
    forall op ∈ O do
        if Pre(op) ⊆ S then
            S ← S ∪ Add(op)
        end
    end
    listo ← (x =| S |)
end
```

Salida

Una vez realizados estos procesos, se obtiene el problema STRIPS, el cual comprende: Una lista de operadores instanciados con 3 conjuntos de átomos asociados: $Pre(op)$, $Add(op)$ y $Del(op)$; un conjunto de átomos y la situación inicial y final como una lista de átomos.

5.1.4. Generador del problema de búsqueda

Este subsistema tiene como objetivo traducir el problema STRIPS a un problema de búsqueda heurística.

Modelo de estados

Como se vio en la sección 2.2, la representación STRIPS define modelos de estados que pueden ser resueltos como un problema de búsqueda heurística. En la implementación, creamos un conjunto de clases abstractas, las cuales describen un modelo de estados.

Estado El principal componente de un modelo de estados, es el estado mismo. La representación del mismo dependerá del tipo de modelo que se desee utilizar. Por ejemplo, la implementación para un estado del modelo STRIPS es un conjunto de enteros, de $1 \dots N$, donde N es el número de átomos. Cada entero en este intervalo está asociado a un átomo.

Operadores Representa las transiciones entre estados. Al igual que los estados, su estructura depende del tipo de modelo que se desee utilizar. En la implementación del modelo STRIPS, un operador consta de tres listas de enteros, que representan: La precondición, la lista de átomos a agregar y la lista a eliminar. Los efectos de su aplicación son ejecutados por la interfaz de búsqueda.

Interfaz de búsqueda Representa las operaciones de búsqueda. Esta clase es instanciada para la búsqueda *hacia adelante* y *en regresión*, ya que ambas tienen comportamientos diferentes.

- **estado** $estadoInicial()$: Retorna el estado inicial del problema.
- **booleano** $esGoal(s)$: Indica si el estado s es objetivo.
- **lista de operadores** $operadoresAplicables(s)$: Retorna una lista de operadores que representa el conjunto $A(s)$, es decir, los operadores que pueden aplicarse en el estado s,
- **estado** $aplicarOperador(s, op)$: Retorna el resultado de aplicar el operador op en el estado s. Este método describe el comportamiento de la función de transición $f(op, s)$.

Heurística

La heurística, como fue explicado anteriormente, es un elemento de suma importancia para el problema de búsqueda. Para representarla, se utilizó una clase abstracta llamada $Heuristica$ que posee un único método: $valor(s)$, el cual retorna el valor de la heurística para el estado s. La implementación dependerá totalmente de la heurística que se desee utilizar. Los detalles de implementación de cada una serán descritos en la proxima sección.
Las heurísticas soportadas por el planificador son:

- Heurística cero: El valor es 0, para todo estado.
- Heurísticas de relajación del problema: Se pueden utlizar h^1, $h^1 - plus$ y h^2. Recordemos que $h^1 - plus$ es una versión no-admisible de la heurística h^1.
- Heurísticas de bases de datos de patrones.

Salida

La salida de esta fase del sistema es el problema de búsqueda, el cual se compone del modelo de estados y la heurística. Estos elementos servirán de entrada para la última fase del sistema, y conducirán a la resolución del problema de planificación.

5.1.5. Resolución de la búsqueda

En la última etapa de la resolución del problema, se tienen todos los elementos necesarios para realizar la búsqueda heurística. Para ello se utiliza el algoritmo A^*, el cual haciendo uso del modelo de estados (representado por la interfaz de búsqueda) y la heurística, halla el camino mínimo que representará la solución del problema de planificación.

5.2. Heurísticas

En el capítulo anterior se explicaron las heurísticas utilizadas en este proyecto y, en el caso de las heurísticas de bases de datos de patrones, los diferentes métodos de selección de patrones. En esta sección se explicarán los detalles de implementación relacionados a estos elementos.

5.2.1. Heurísticas de relajación

En el proyecto, fueron implementadas las heurísticas h^1 y h^2. En *HSP2.0*, al igual que el planificador desarrollado en este trabajo, se calcula la heurística h^2 utilizando un algoritmo muy similar a Bellman-Ford [3] en el cual se asigna un costo estimado a cada par de átomos y luego se aplican actualizaciones consecutivas hasta que los costos convergen en su valor real [16]. Este método está descrito en el algoritmo 4.

Para entender este algoritmo, es necesario reescribir la ecuación 4.1 para el caso $m = 2$.

$$
\begin{aligned}
h^2(\{p\}) = & \min_{\{a \mid p \in add(a)\}} (h^2(pre(a)) + cost(a)) \\
h^2(\{p,q\}) = & \min \left(\begin{array}{l} \min_{\{a \mid p,q \in add(a)\}} (h^2(pre(a)) + cost(a)), \\ \min_{\{a \mid p \in add(a),\ q \notin del(a)\}} (h^2(pre(a) \cup \{p\}) + cost(a)) \\ \min_{\{a \mid q \in add(a),\ p \notin del(a)\}} (h^2(pre(a) \cup \{q\}) + cost(a)) \end{array} \right)
\end{aligned}
\tag{5.1}
$$

Como podemos observar, el algoritmo 4 asigna un costo inicial a cada par de átomos y luego aplica actualizaciones consecutivas, utilizando la ecuación 5.1, hasta que los costos convergen en su valor real.

La heurística h^2 cumple un papel muy importante en este proyecto. Si bien el objetivo del mismo es el estudio de métodos para la construcción de heurísticas basadas en bases de datos de patrones, las heurísticas h^1 y h^2 son utilizadas en muchas etapas de estos procesos. Por ejemplo, la heurística h^2 es utilizada para hallar conjuntos de invariantes, específicamente mutexes (véase sección 3.1.2), utilizando el siguiente teorema:

Teorema 5.2.1 *Sean a y b dos átomos del problema. Si* $h^2(\{a,b\}) = \infty$*, entonces* $\{a,b\}$ *constituye un mutex.*

5.2.2. Heurísticas de bases de datos de patrones

En esta sección discutiremos los detalles de implementación de las heurísticas de bases de datos de patrones.
De acuerdo a lo explicado en la sección 4.2, enumeremos los principales procesos necesarios para obtener las heurísticas PDB:

- **Implementación de la representación en variables multi-valuadas SAS.** Como fue explicado en la sección 4.2.2, basar los patrones en variables multi-valuadas hace más efectivo el uso de la memoria. Por esta razón es necesario *traducir* el problema STRIPS, a un problema donde utilicemos la representación SAS.
- **Construcción de la PDB para un patrón dado**. Este proceso es básico en la utilización de heurísticas de bases de datos de patrones. Consiste, en pocas palabras en: dado un patrón A, construir la tabla que representa la heurística $h^A(s)$, a través de una búsqueda exhaustiva en el problema proyectado sobre A (véase sección 4.2.1).

Algoritmo 4: Algoritmo para cálculo de h^2

Entrada: $S = \langle A, O, I, G \rangle$ el problema STRIPS sobre el cual queremos calcular h^2
forall $p \in A$ **do**
  if $p \in I$ **then**
    $T(\{p\}) \leftarrow 0$
  else
    $T(\{p\}) \leftarrow \infty$
  end
end
forall $p, q \in A$ **do**
  if $p, q \in I$ **then**
    $T(\{p,\ q\}) = 0$
  else
    $T(\{p,\ q\}) = \infty$
  end
end
$cambio \leftarrow$ TRUE
while *cambio* **do**
  $cambio \leftarrow$ FALSE
  forall $a \in A$ **do**
    $c_1 \leftarrow eval(pre(a))$
    forall $p \in add(a)$ **do**
      $actualiza(\{p\},\ c1 + cost(a))$
      forall $q \in add(a),\ q \neq p$ **do**
        $actualiza(\{p, q\}, c1 + cost(a))$
        forall $r \notin del(a),\ r \neq p$ **do**
          $c_2 \leftarrow eval(pre(a) \cup \{q\})$
          $actualiza(\{p, r\}, c2 + cost(a))$
        end
      end
    end
end
actualiza(s, v){
  if $T(s) > v$ **then** $T(s) \leftarrow v$, $cambio \leftarrow TRUE$
eval(S){
  $ret \leftarrow 0$
  forall $p, q \in S$ **do** $ret \leftarrow max(ret, T(p, q))$
  retorna ret
}

- En la sección 4.2.4, el esquema general de selección de patrones, sobre el cual se basan todos los métodos de selección estudiados en este proyecto, parte de **conjuntos aditivos de variables**. En la implementación, fue necesario desarrollar un método para determinar dichos conjuntos.

Traducción de STRIPS a SAS

En su estudio, Edelkamp [10] propone como primer paso para la construcción de una PDB un análisis sobre el dominio y, como resultado de éste, una partición de los átomos en conjuntos mutuamente exclusivos, es decir, para cada conjunto P en la partición, a lo sumo un átomo pertenece a todo estado alcanzable. Esta partición es representada por una estructura que llamaremos *SASMAP*, la cual definirá las variables del modelo SAS. Veamos un ejemplo:

Ejemplo SASMAP: Blocksworld Consideremos un problema de 3 bloques: A, B y C. Los átomos del problema están señalados en la sección 2.1.1. Una partición en conjuntos mutuamente exclusivos sería:

$$\begin{aligned} G_1 &= \texttt{(on-table A)}, \texttt{(on A B)}, \texttt{(on A C)} \\ G_2 &= \texttt{(on-table B)}, \texttt{(on B A)}, \texttt{(on B C)} \\ G_3 &= \texttt{(on-table C)}, \texttt{(on C A)}, \texttt{(on C B)} \\ G_4 &= \texttt{(clear A)} \\ G_5 &= \texttt{(clear B)} \\ G_6 &= \texttt{(clear C)} \end{aligned} \tag{5.2}$$

Podemos observar que, de los átomos en los conjuntos G_1, G_2, y G_3, a lo sumo uno de ellos puede estar presente en todo estado válido ya que un bloque (A, B ó C) no puede estar sobre la mesa y sobre otro bloque al mismo tiempo. Los átomos restantes conforman cada uno un conjunto, para completar la partición. De aquí se deriva la representación de estados SAS, en la cual cada estado está descrito por una lista de asignación a variables, en vez de un conjunto de átomos como en STRIPS. En el ejemplo anterior, asociemos a cada grupo una variable, y los elementos del mismo serán los posibles valores que puede tomar. El estado $s = \{\texttt{(on-table A)}, \texttt{(on B A)}, \texttt{(on-table C)}, \texttt{(clear B)}, \texttt{(clear C)}\}$ en STRIPS, sería representado en SAS como: $s = \langle V_1 = \texttt{(on-table A)}, V_2 = \texttt{(on B A)}, V_3 = \texttt{(on-table C)}, V_4 = \varnothing, V_5 = \texttt{(clear B)}, V_6 = \texttt{(clear C)}\rangle$. Como podemos apreciar, para completar la representación, debemos agregar un elemento a cada grupo, el elemento nulo $\varnothing$, para indicar que la variable no tiene valor, como sucede en el caso de V_4. Nuestro *SASMAP* para el problema Blocksworld sería el siguiente:

$$
\begin{aligned}
V_1 \rightarrow &\ \varnothing, \texttt{(on-table A)}, \texttt{(on A B)}, \texttt{(on A C)} \\
V_2 \rightarrow &\ \varnothing, \texttt{(on-table B)}, \texttt{(on B A)}, \texttt{(on B C)} \\
V_3 \rightarrow &\ \varnothing, \texttt{(on-table C)}, \texttt{(on C A)}, \texttt{(on C B)} \\
V_4 \rightarrow &\ \varnothing, \texttt{(clear A)} \\
V_5 \rightarrow &\ \varnothing, \texttt{(clear B)} \\
V_6 \rightarrow &\ \varnothing, \texttt{(clear C)}
\end{aligned}
\tag{5.3}
$$

En la implementación, enumeramos los m posibles valores de cada variable desde $0 \ldots m-1$, siendo el 0 el valor $\varnothing$. Si hubieran n variables, un estado estaría representado como una lista de n enteros $(v_1, v_2, \ldots, v_n)$ donde v_i es el valor de la variable V_i. De la misma forma, un operador SAS está representado por tres listas de n enteros, las listas corresponden a la precondición, el conjunto de átomos a agregar y a eliminar. Los enteros de estas listas $(v_1, v_2, \ldots, v_n)$, al igual que en los estados, representan los valores para cada variable V_i.
Cabe destacar que el *SASMAP* no debe ser necesariamente disjunto, es decir, pueden existir átomos que pertenezcan a dos o mas variables. Un ejemplo de *SASMAP* no disjunto para Blocksworld:

$$
\begin{aligned}
V_1 \rightarrow &\ \varnothing, \texttt{(on-table A)}, \texttt{(on A B)}, \texttt{(on A C)} \\
V_2 \rightarrow &\ \varnothing, \texttt{(on-table B)}, \texttt{(on B A)}, \texttt{(on B C)} \\
V_3 \rightarrow &\ \varnothing, \texttt{(on-table C)}, \texttt{(on C A)}, \texttt{(on C B)} \\
V_4 \rightarrow &\ \varnothing, \texttt{(clear A)}, \texttt{(on B A)}, \texttt{(on C A)} \\
V_5 \rightarrow &\ \varnothing, \texttt{(clear B)}, \texttt{(on A B)}, \texttt{(on C B)} \\
V_6 \rightarrow &\ \varnothing, \texttt{(clear C)}, \texttt{(on A C)}, \texttt{(on B C)}
\end{aligned}
\tag{5.4}
$$

La condición de mutua exclusividad de los elementos de un mismo conjunto se sigue cumpliendo y el número de variables se mantiene igual.
Edelkamp [10] propone que la construcción de una PDB debe ser automática e independiente del dominio. Por ende, la definición de el *SASMAP*, como primer paso para la elaboración de una PDB, debe ser sistemática, a través de un análisis del dominio. En este proyecto se implementó un método para determinar las variables del *SAPMAP* utilizando los mutexes.

Un mutex, como vimos en la sección 3.1.2, es un tipo de invariante donde el conjunto sólo contiene dos elementos. Pero también, en un invariante con más de dos átomos, todo par de átomos del conjunto son mutex. Una forma muy simple de detectar mutexes para un problema, es utilizando la heurística h^2, como vimos en el teorema 5.2.1. De aquí se deriva nuestro procedimiento para elaborar un *SASMAP*,

primero calculamos la heurística h^2, luego una partición, o una división no-disjunta según fuese el caso, de átomos donde en cada conjunto, para cada par de átomos $\{p, q\}$, se cumpla $h^2(\{p, q\}) = \infty$.

Construcción de la partición de átomos Luego de obtener la heurística h^2, proponemos el siguiente procedimiento para hallar una partición. Inicialmente, construimos un grafo $G = (V, E)$, no dirigido, donde:

- V, el conjunto de vértices, es igual al conjunto de átomos del problema.
- E, el conjunto de pares de vértices, es tal que: $\forall\ e = \{a, b\},\ e \in E \mid h^2(a, b) = \infty$. Es decir, colocamos una arista entre a y b sin son mutex.

A continuación, utilizamos un algoritmo para hallar cliques maximales para obtener conjuntos de vertices (átomos) que conformen un invariante. De forma análoga, podemos utilizar el grafo dual, con un algoritmo para hallar un conjunto independiente maximal, obteniendo el mismo resultado. Los algoritmos implementados fueron Luby [27] y Bron-Kerbosch [9]. El primero, un algoritmo para calcular un conjunto independiente maximal, es randomizado, puede realizarse en paralelo y es considerado eficiente. El segundo, para hallar un clique maximal, utiliza backtracking combinado con la técnica branch-and-bound. El mismo es considerado hoy en día como uno de los más eficientes. Dado que ambos algoritmos hallan un conjunto maximal (independiente o clique), para hallar una partición es suficiente realizar varias llamadas a los algoritmos antes mencionados y, a medida que se van obteniendo los conjuntos maximales, los vértices que los conforman son eliminados del grafo, así como las aristas que los mencionan. Este proceso se realiza hasta que el grafo sea vacío. Cabe destacar que en la experimentación se utilizó el algoritmo Bron-Kerbosch para la construcción del *SASMAP*.

Construcción de una PDB

Una vez obtenido el *SASMAP*, un patrón consistirá en un conjunto de variables, y la heurística final en un conjunto de patrones. Como vimos anteriormente, el valor almacenado para cada estado abstracto s en una PDB es el costo camino mínimo desde s hasta un estado abstracto final. Para ello se propone utilizar una búsqueda exhaustiva en el espacio de regresión, definido en la sección 2.2.1. Antes de realizar la búsqueda, es necesario proyectar el problema sobre el patrón, es decir, descartamos de los operadores SAS, del estado inicial y del objetivo, aquellas variables que no estén incluídas en P. El siguiente paso es realizar la búsqueda exhaustiva, utilizando el algoritmo Dijkstra [30], para calcular el costo mínimo de alcanzar el objetivo desde todos los estados alcanzables.

Asignaciones completas y parciales Un detalle importante para la implementación es la diferencia entre las asignaciones completas y las parciales. Una asignación se dice completa si toda variable tiene un valor, es decir: si existen n variables, el estado representado consiste en n átomos. La diferencia principal entre ambos tipos es la forma en la cual se calculan sus valores en la PDB. El valor de una asignaciones parciales es el mínimo entre todas las posibles completaciones [16]. En la implementación utilizamos Dijkstra para los valores de las asignaciones completas, combinado con un algoritmo de programación dinámica para computar el costo mínimo de las asignaciones parciales. Este enfoque es propuesto por Haslum [16].

Determinación de conjuntos aditivos de variables

Como vimos anteriormente, la aditividad de patrones nos permitirá obtener heurísticas más acertadas dado que podemos sumar valores de las PDB's sin riesgo de perder la admisibilidad. Una definición formal de la aditividad de patrones es la siguiente:
Dos patrones A y B son aditivos si y sólo si:

$$add(a) \cap A = \varnothing \quad \vee \quad add(a) \cap B = \varnothing \quad \forall \text{ acción } a \tag{5.5}$$

es decir, ninguna acción agrega átomos de ambos conjuntos.
Para determinar una partición del conjunto de variables en conjuntos aditivos, nos topamos con un problema similar al de determinar la estructura $SASMAP$ de la sección 5.2.2. En este proyecto, el método utilizado consiste en plantear un grafo $G = (V, E)$ donde los vértices en V son las variables y se agrega una arista entre dos variables si se cumple la condición de aditividad. Luego aplicamos uno de los algoritmos para hallar cliques maximales o conjuntos maximales independientes. El resultado será una partición P del conjunto de variables, donde cada conjunto $C \in P$ cumple que $\forall A, B \in C | A \ y \ B$ son independientes.

Capítulo 6

EXPERIMENTACION

Los diferentes métodos de selección de patrones son comparados en 3 dominios diferentes: *Blocksworld*, *15-puzzle* y *Logistics*. Utilizamos bases de datos de patrones con restricciones, utilizando la heurística h^2 para determinar el conjunto C de invariantes. El límite de tamaño de cada tabla fue 2 millones de entradas para todos los dominios y 1GB de memoria para todo el proceso de planificación. En el caso de la estrategia *Gap set*, se estableció el tamaño de la muestra de estados en 250 para *blocksworld*, y 150 para *15-puzzle* y *Logistics*.

6.1. Blocksworld

La descripción de este dominio fue explicada en la sección 2.1.1. Para los experimentos en este dominio se utilizó un SASMAP no-disjunto, donde se definen dos tipos de variables para cada bloque:

- $pos(X)$: describe sobre que objeto se encuentra el bloque X.
- $top(X)$: describe que se encuentra sobre X.

Las variables están definidas de la siguiente forma:

```
pos(X) = {(on-table X) (on X Y)} para todo bloque Y
top(X) = {(on-table Y) (on Y X)} para todo bloque Y
```

Como es fácil de observar, el número total de variables es dos veces el número de bloques. Las instancias que fueron probadas son de 10,11 y 12 bloques, con un total de 60 experimentos.

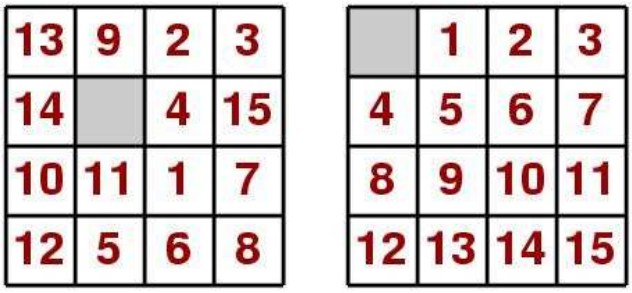


Figura 6.1: Ejemplo de estados en el 15-puzzle. A la izquierda, un estado aleatorio. A la derecha, el objetivo del juego

6.2. 15-puzzle

El *15-puzzle* es un juego de tipo rompe-cabezas que consiste en una retícula de 4x4 donde en se disponen 15 fichas, enumeradas desde 1 a 15, cada una en una casilla. El objetivo del juego es, a partir de una configuración inicial del tablero, llegar a un ordenamiento final de las fichas (estado objetivo). Los movimientos válidos son el desplazamiento de las fichas adyacentes a la casilla vacia, donde Este dominio, al igual que el 8-puzzle (3×3) y el 24-puzzle (5×5) [26] han sido estudiados y utilizados en muchos temas de Inteligencia Artificial, sobre todo aquellos relacionados con búsqueda heurística, dada su dificultad y complejidad. Para la especificación en STRIPS se utiliza el predicados `(at t x y)` donde t representa una ficha ($t \in \{t1, t2, \ldots, t15\}$) y (x, y) ($1 \leq x, y \leq 4$) para describir la posición de una ficha en el tablero. En este problema, al igual que el anterior, se especifica el SASMAP donde se define, pricipalmente, una variable por cada ficha.

```
pos(T) = {(at T x y}} para (x,y) del tablero
```

Como podemos observar, los átomos pertenecientes a una variable $pos(T)$ son mutuamente exclusivos ya que una ficha no puede ocupar dos posiciones en el tablero. Las instancias del problema que fueron utilizadas son un subconjunto de la colección de Korf [24].

6.3. Logistics

El dominio *Logistics* consiste en el transporte de paquetes entre almacenes a través de camiones o aviones. Los paquetes se encuentran en almacenes dentro de ciudades. Una ciudad también puede poseer aeropuertos a través de los cuales, mediante el traslado en avión, se puede alcanzar otra ciudad. Los camiones únicamente realizan traslados dentro de una misma ciudad y sólo pueden llevar un paquete a

la vez. Existen dos versiones de este dominio que se diferencian por poseer o no un sistema de tipos. En la versión sin tipos, se introducen un conjunto de átomos para aplicar las restricciones sobre las objetos, lo cual genera estados de mayor longitud. En este proyecto aprovechamos los beneficios del sistema de tipos para maximizar la eficiencia en la ejecución del planificador. Una de las características de este dominio es la gran longitud de los planes, así como la gran cantidad de variables que se obtienen. Para este problema utilizamos los algoritmos descritos en la sección 5.2.2. Las variables obtenidas son de la siguiente forma:

- $PosPaquete(X)$: Agrupa los átomos que describen la posición del paquete X. Un paquete sólo puede estar en un lugar a la vez.
- $PosCamion(T)$: Indica la posición del camión T.
- $PosAvion(A)$: Indica la posición del avión A.

Como podemos observar, cada variable representa un invariante, que está asociado a la propiedad de que un objeto (paquete, camión o avión) no puede estar en dos lugares a la vez. Cabe destacar que esta partición de átomos en variables es obtenida de forma automática, utilizando el método descrito en la sección 5.2.2. Las 55 instancias de este problema fueron obtenidas mediante un programa creado por Albert Ludwigs [20] para generar problemas de forma aleatoria. Cabe destacar que se realizaron modificaciones al programa antes mencionado para obtener instancias de la versión con tipos.

Capítulo 7

RESULTADOS Y CONCLUSIONES

Habiendo visto la descripción de los dominios que fueron utilizados en la experimentación, se presentarán los resultados para cada uno de los métodos de selección de patrones. Para evaluar la calidad y la efectividad de cada estrategia, se tomaron en cuenta tres factores:

1. **El número de nodos expandidos** por el algoritmo A^* durante la búsqueda. Para determinar que la efectividad al guiar la búsqueda.
2. **El tiempo de construcción de la heurística.**
3. **La ponderación de los conflictos** entre patrones. Utilizando el número de empates sobre el número total de iteraciones para indicar que tan determinística es la selección.
4. **Porcentaje de problemas resueltos** para cada dominio.

7.1. Número de nodos expandidos

En la figura 7.1 se muestra la distribución acumulativa del tiempo de cálculo de la heurística para cada estrategia de selección de patrones, en cada uno de los dominios. El eje de las coordenadas (Número de nodos expandidos) es logarítmico. La tabla 7.1 muestra el número promedio de nodos expandidos, en cada uno de los dominios, para cada estrategia. Podemos observar que en el dominio *Blocksworld*, las estrategias *Gap set* y *Selección incremental* demuestran desempeños similares, a pesar de las diferencias en el número promedio de nodos expandidos, motivadas principalmente a un algunas instancias particulares. La estrategia *Gap at init* no demuestra ser eficiente en este dominio. Para el dominio *15-puzzle*, el *Gap set* y la *Selección incremental* muestran un comportamiento curioso, la primera supera a la segunda en el 50 % de las instancias, mientras que en el otro 50 % ocurre lo contrario, lo cual sugiere un desempeño promedio similar. Los resultados de la tabla 7.1 respaldan lo anterior mostrando una diferencia de

apenas 300 nodos en el número promedio para ambas estrategias. La estrategia *Gap at init* nuevamente demuestra un desempeño pobre. En el dominio *Logistics*, *Gap set* y *Gap at init* muestran resultados muy similares, siendo la primera la más eficiente al tomar en cuenta el número promedio de nodos expandidos. La estrategia *Selección incremental* no demuestra ser eficiente en este dominio. Las razones de este resultado serán explicadas más adelante, al analizar otros factores.

Dominio	**Sel.Incremental It.**	**Gap at init**	**Gap set**
Blocksworld-10	10496.44	28562.44	22806.24
Blocksworld-11	22597.11	67047.88	55416.04
Blocksworld-12	118082.26	96636.33	88314.05
15-puzzle	156983.33	346036.65	156609.58
Logistics	77194.31	47782.52	34494.17

Tabla 7.1: Número de nodos expandidos promedio.

7.2. Tiempo de creación de la heurística

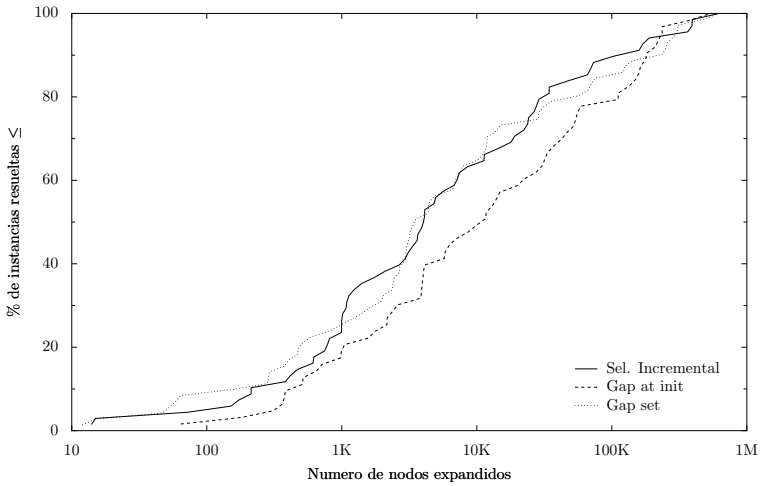
La tabla 7.2 nos muestra los tiempos promedio de creación de la heurística y la gráfica 7.2 la distribución acumulativa utilizando cada uno de los métodos de selección, para cada dominio. En los dominios *Blocksworld* y *15-puzzle*, la estrategia *Gap at init* muestra mejores tiempos. En el dominio *Logistics*, la *Selección incremental* muestra mejores tiempos, motivado principalmente a una selección arbitraria de patrones en todas las instancias, como se verá en la próxima sección. En todos los dominios, la estrategia *Gap set* demuestra requerir mucho más tiempo.

Dominio	**Sel.Incremental It.**	**Gap at init**	**Gap set**
Blocksworld-10	175.5s	202.68s	328.76s
Blocksworld-11	67.68s	20.58s	86.72s
Blocksworld-12	168.48s	28.12s	271.12s
15-puzzle	848.43s	333.79s	1821.25s
Logistics	166.33s	339.04s	4904.37s

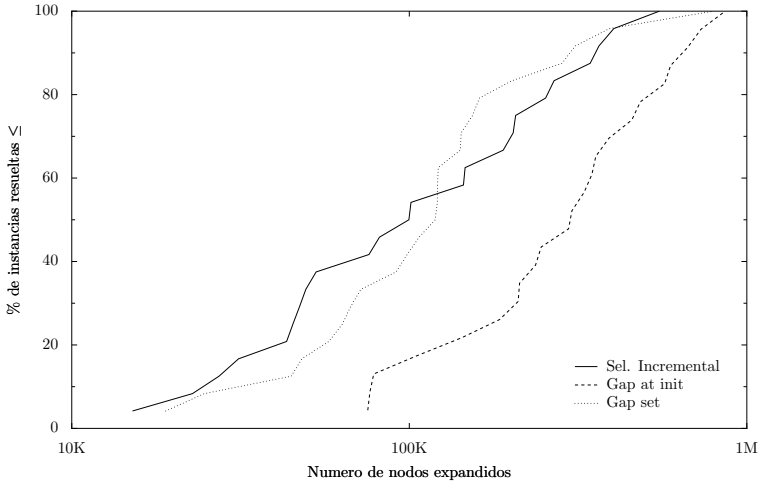
Tabla 7.2: Tiempos promedio de creación de la heurística.

Efecto del tamaño de las tablas sobre el tiempo de construcción de la heurística

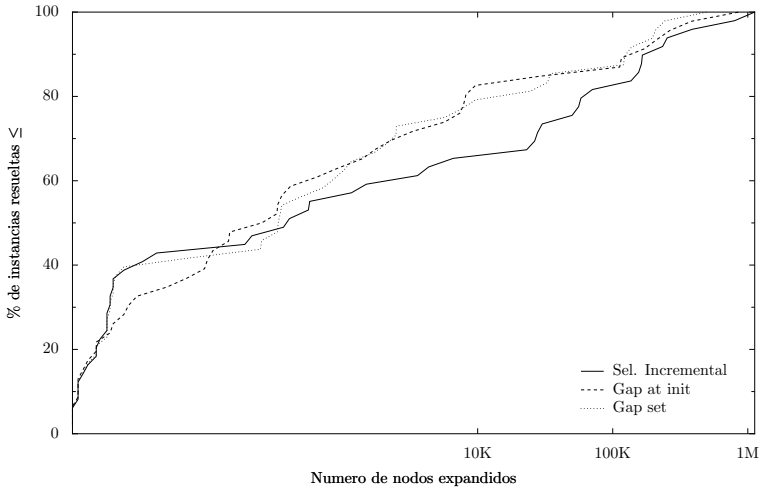
En la tabla 7.2 podemos apreciar la gran cantidad de tiempo que consume la estrategia *Gap set* en comparación a las demás. Considerando la implementación de esta estrategia, planteada en la sección 4.2.4, este tiempo podría ser atribuido principalmente a las numerosas llamadas al algoritmo IDA^*. Pero observando los tiempos para la estrategia de selección incremental, podemos notar que existen instancias en el dominio *15-puzzle* para las cuales el tiempo de construcción de la heurística fue mayor a 5000



Blocksworld

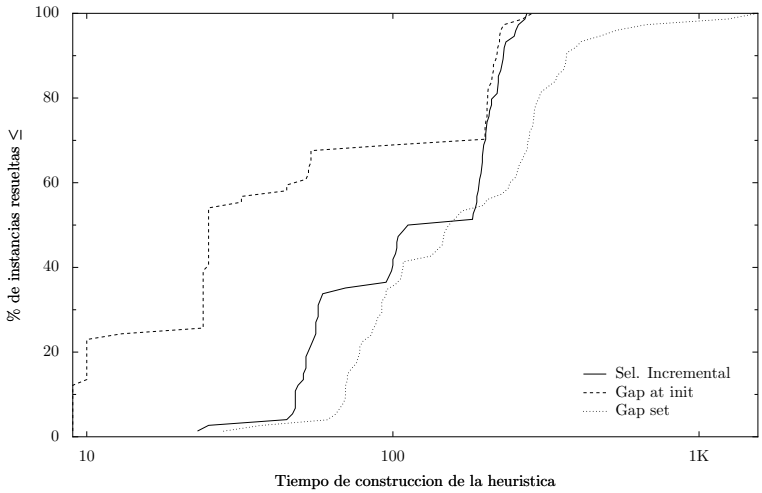


15-puzzle

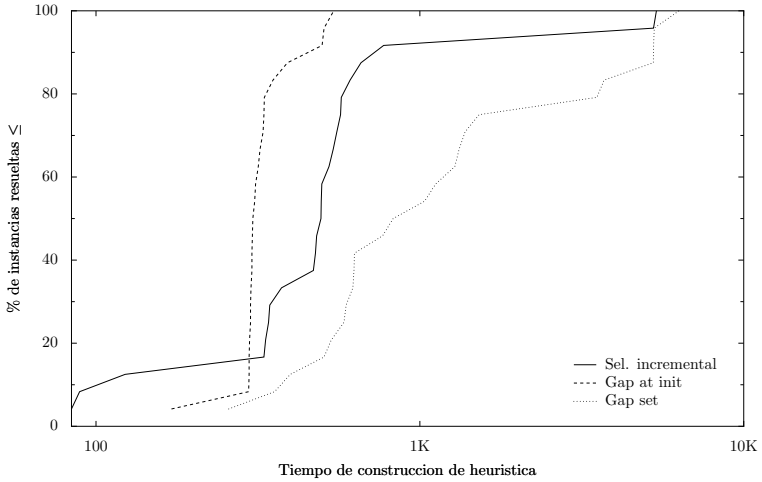


Logistics

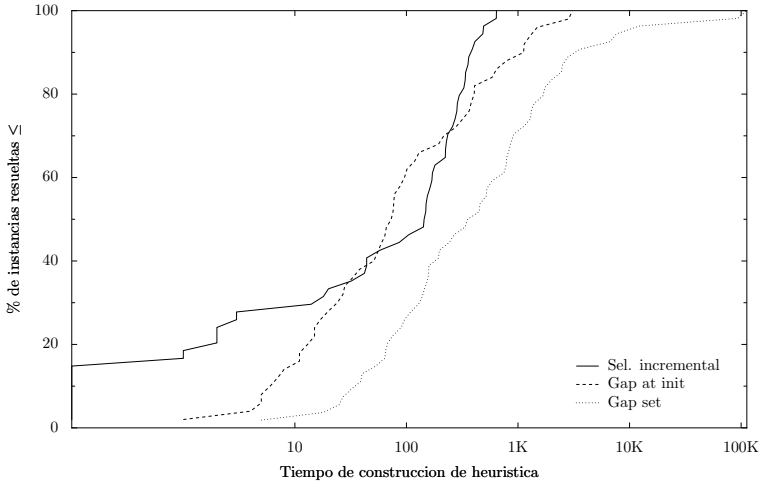
Figura 7.1: Distribución acumulativa del número de nodos expandidos en la búsqueda A* utilizando las diferentes estrategias de selección de patrones



Blocksworld



15-puzzle



Logistics

Figura 7.2: Distribución acumulativa del tiempo de construcción de la heurística

segundos. La varianza de este factor para cada una de las estrategias nos hace plantear la idea que, si bien la estrategia *Gap set* consume en general más tiempo, es la construcción de las tablas el factor determinante del tiempo de elaboración de la heurística. Consideremos el siguiente ejemplo:

Ejemplo de construcción de heurística. *15-puzzle* En la sección 6.2 se describieron las variables que fueron establecidas para este dominio. Cuando se calculan los conjuntos aditivos maximales, se obtiene un conjunto al cual pertenecen todas las variables $pos(X)$ para todo $X \in \{t_1, \ldots, t_{15}\}$, es decir, tenemos un conjunto aditivo de 15 patrones , cada uno con tamaño 16. Recordemos que fue establecido un límite de 2,000,000 entradas para cada PDB. Para simplicar la descripción, denotaremos $A = pos(t_1), B = pos(t_2), \ldots, O = pos(t_{15})$.

$$\begin{aligned} R_1 = &\quad \{\{A \cup B \cup C\}, \{D \cup E \cup F\}, \{G \cup H \cup I\}, \{J \cup K \cup L\}, \{M \cup N \cup O\}\} \\ R_2 = &\quad \{\{A \cup B \cup C \cup D \cup E\}, \{F \cup G \cup H \cup I \cup J\}, \{K \cup L \cup M \cup N \cup O\}\} \end{aligned} \tag{7.1}$$

Como puede observarse, ambos resultados están dentro del límite de 2,000,000 de entradas por PDB, y son finales ya que no puede realizarse otra unión de patrones. El número de entradas, en el peor caso, que debieron calcularse para llegar a estos resultados son:

$$\begin{aligned} R_1 = &\quad 16 * (16) + 5 * (16^2) + 5 * (16^3) = 22,016 \\ R_2 = &\quad 16 * (16) + 3 * (16^2) + 3 * (16^3) + 3 * (16^4) + 3 * (16^5) = 3,355,648 \end{aligned}$$

En la figura 7.3 podemos comparar los tiempos de cálculo de las estrategias *Gap set* y *Selección incremental* para el dominio *15-puzzle* disminuyendo el límite del tamaño de las PDBs. Se puede notar la gran diferencia de tiempos existente en el *Gap set*. El nuevo tiempo del *Gap set* incluso mejora el resultado anterior de la estrategia *Selección incremental*. En cuanto a tiempos promedio, el *Gap set* se redujo a más de la mitad (636.21s) al disminuir el tamaño. Esta diferencia respalda la hipótesis de que el factor determinante, como ejemplificamos anteriormente, es la construcción de las tablas.

7.3. Ponderación de conflictos

En las todas las estrategias utilizadas es común que ocurran empates en algunas iteraciones, lo cual conlleva, en esas iteraciones, a una selección arbitraria de los patrones a unir. En la tabla 7.3 se muestra el porcentaje promedio de iteraciones en las cuales ocurrieron empates, sobre el número total de iteraciones del dominio.

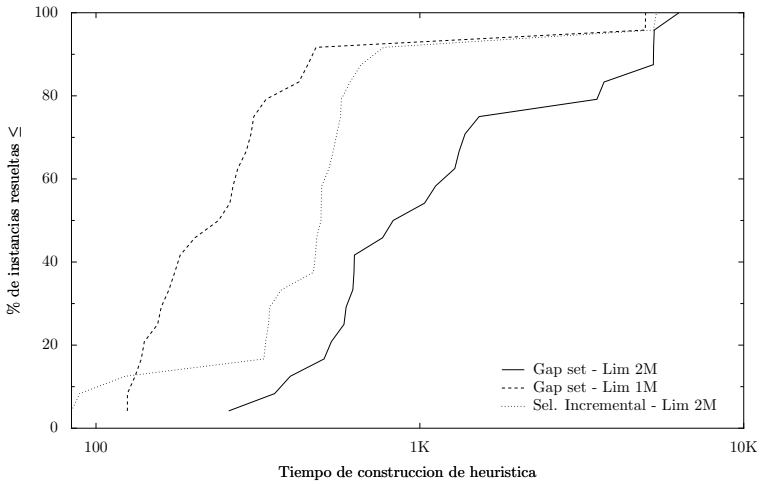


Figura 7.3: Distribución acumulativa del tiempo de construcción de la heurística en el dominio *15-puzzle* al establecer el límite de tamaño de cada tabla en 1 millón de entradas

Dominio	**Sel.Incremental It.**	**Gap at init**	**Gap set**
Blocksworld-10	63 %	82 %	36 %
Blocksworld-11	68 %	89 %	33.5 %
Blocksworld-12	64.89 %	87.1 %	34.2 %
15-puzzle	91.92 %	98.08 %	83 %
Logistics	100 %	91.15 %	72.23 %

Tabla 7.3: Porcentaje promedio de iteraciones con empates.

Analizando estos resultados, podemos concluir que la estrategia *Gap set* es la más discriminatoria en todos los dominios considerados. La principal razón es el rango en el que varían los pesos de los conflictos. Tomemos como ejemplo el dominio *Blocksworld*, donde la longitud promedio de los planes es de 15 acciones. Para la estrategia *selección incremental*, recordemos que el peso de un conflicto es el valor h^1 del conjunto de átomos inconsistente. El rango, muy exagerado de estos valores, sería entre 0 y la longitud del plan (15). Para la estrategia *Gap init*, en la experimentación se observó que las diferencias, en promedio, suelen ser $h^{A\cup B}(s) - h^A(s) - h^B(s) \leq 2$. Esto indica que los conflictos están poderados entre 0 y 2. Para el *Gap set*, con una muestra $|S| = 250$, estaríamos considerando un rango entre 0 y 750. Estas diferencias considerables de rangos, y sus implicaciones, se reflejan claramente en los experimentos. Otra observación a realizar es el desempeño de la estrategia *Selección incremental* en el dominio *Logistics*. Un valor de 100 % de empates indica que la selección de los patrones fue totalmente arbitraria.

Relación entre el tiempo de creación de la heurística y la ponderación de los conflictos

Para explicar la relación entre estos dos factores es necesario entender algunos detalles de la implementación de las estrategias de selección. Para todas las estrategias, el proceso comienza con una lista

de patrones. A medida que van transcurriendo las iteraciones y las uniones son realizadas, los nuevos patrones conformados son agregados al final de esta lista. En el momento de seleccionar el conflicto, se toma el primero de ellos en la lista que cumpla con el valor máximo. Estos dos detalles en conjunto provocan en los problemas donde suceden numerosos empates que se conformen varios patrones de poco tamaño, similares al resultado R_1 del ejemplo 7.1. Esta situación disminuye considerablemente el tiempo de construcción de la heurística. Bajo esta premisa, se puede explicar el hecho de que la estrategia *Gap set* sea la que tenga menor porcentaje promedio de empates y su construcción lleve más tiempo, en contraste a la estrategia *Gap at init*, la cual tiene mayor número de empates y consume menos tiempo. Un ejemplo claro de la relación de estos dos factores es el resultado para el dominio *Logistics*, donde la *Selección incremental* tuvo 100 % de iteraciones con empates y un tiempo mucho menor de creación de la heurística, en contraste con la estrategia *Gap set*.

7.4. Porcentaje de problemas resueltos

En la tabla 7.4 podemos apreciar el porcentaje de problemas resueltos por cada una de las estrategias en los dominios antes mencionados. Se puede observar que la estrategia *Gap at init* resuelve un número igual o menor que las otras en todos los dominios. El *Gap set* demuestra resolver más problemas que la *Selección incremental* en todos los dominios, a excepción de *Logistics*. Sin embargo, la diferencia es sólo una instancia y como se vio anteriormente, la *Selección incremental* demuestra ser arbitraria en este dominio.

Dominio	**Sel.Incremental It.**	**Gap at init**	**Gap set**
Blocksworld-10	100 %	100 %	100 %
Blocksworld-11	96 %	96 %	96 %
Blocksworld-12	76 %	56 %	88 %
15-puzzle	100 %	96 %	100 %
Logistics	91 %	85 %	89 %

Tabla 7.4: Porcentaje de problemas resueltos.

7.5. Conclusiones y recomendaciones

Los resultados de los experimentos muestran que, en los tres dominios considerados, el nuevo método de selección de patrones propuesto, *Gap set*, es competitivo con la *selección incremental* propuesta por [17]. Luego de un análisis de cada uno de los aspectos, se pueden realizar algunas observaciones interesantes. En primer lugar, si bien el tiempo de construcción de la heurística difiere mucho en todos los métodos, gran parte de este se atribuye a la construcción de las tablas, no al proceso de selección.

En segundo lugar, dado que la selección de cuales patrones se deben unir cuando ocurren empates es significante [16], un buen método de selección debe ser determinístico. Los resultados muestran que en la *selección incremental*, ocurren empates en la mayoría de las iteraciones. Un resultado interesante es el porcentaje de empates de esta estrategia en el dominio *Logistics*. En las 55 instancias generadas de forma aleatoria, en las todas las iteraciones (19 en promedio) ocurrieron empates. Tomando en cuenta este aspecto, el *Gap set* demostró ser el más determinístico. Tomando en cuenta la eficiencia de las heurísticas para guiar la búsqueda, el *Gap set* demostró resultados uniformes al ser más eficiente en dos de los tres dominios considerados.

7.5.1. Recomendaciones para futuras investigaciones

El análisis de la calidad de las heurísticas obtenidas por los diferentes métodos de selección de patrones señala un área en la cual pueden realizarse futuros estudios, entre los cuales se encuentra el desarrollo de nuevos métodos, o la implementación de mejoras a los ya existentes, para hacer las selecciones más informadas y más determinísticas.

Para el método *Gap set* existe un punto sobre el cual pueden realizarse futuras investigaciones: la selección de la muestra de estados S. En los experimentos se mostró que en las primeras iteraciones, cuando los patrones son de tamaños reducidos, este método suele ser muy determinístico, pero al avanzar el proceso de selección, el tamaño de la muestra (250 para blocksworld y 150 para los otros dominios) es muy reducido en comparación al tamaño de las tablas. Por esta razón, se ponderan muchos conflictos con valor 0. El proceso de construcción del conjunto S en este proyecto, como fue explicado en la sección 4.2.4, es muy simple y poco informado. El desarrollo de nuevos métodos, más informados, para obtener esta muestra puede llevar a obtener mejores heurísticas.

Otro punto de interés son las posibles modificaciones al proceso de general de selección de patrones. Como se pudo observar, los dos métodos propuestos en este proyecto parten de la misma base de la *selección incremental*, es decir, se toma cada conjunto aditivo de variables, se crea un patrón por cada variable y se realizan iteraciones, en cada una de las cuales se seleccionan dos patrones, según algún criterio, y se unen para conformar un nuevo patrón. Una modificación planteada en la sección 4.2.4 explica una variante a este proceso, en la cual se pueden considerar tríos de patrones, es decir, ponderar los conflictos para cada trio de patrones mediante la diferencia $h^{A\cup B\cup B)} - h^A(s) - h^B(s) - h^C$. Cabe determinar si esta modificación al esquema general mejora la calidad de las heurísticas de bases de datos de patrones.

REFERENCIAS

[1] A. Aaby. Compiler construction using Flex and Bison, 1996.

[2] C Bäckström and B Nebel. Complexity results for SAS^+ planning. Research Report LiTH-IDA-RR-93-34, Department of Computer and Information Science, Linköping University, Linköping, 1993.

[3] P. Black. Bellman-ford algorithm, in dictionary of algorithms and data structures [online]. http://www.nist.gov/dads/HTML/bellmanford.html (18 de marzo de 2007).

[4] A. Blum and M. Furst. Fast planning through planning graph analysis. *aij*.

[5] A. Blum and M. Furst. Fast planning through planning graph analysis. In C. Mellish, editor, *Proc. 14th International Joint Conf. on Artificial Intelligence*, pages 1636–1642, Montreal, Canada, 1995. Morgan Kaufmann.

[6] B. Bonet and H. Geffner. Heuristic search planner 2.0. *Artificial Intelligence Magazine*, 22(3):77–80, Fall 2001.

[7] B. Bonet and H. Geffner. Planning as heuristic search. *Artificial Intelligence*, 129(1–2):5–33, 2001.

[8] B. Bonet, G. Loerincs, and H. Geffner. A robust and fast action selection mechanism for planning. In B. Kuipers and B. Webber, editors, *Proc. 14th National Conf. on Artificial Intelligence*, pages 714–719, Providence, RI, 1997. AAAI Press / MIT Press.

[9] C. Bron and J. Kerbosch. Algorithm 457: finding all cliques of an undirected graph. *Commun. ACM*, 16(9):575–577, 1973.

[10] S. Edelkamp. Planning with pattern databases. In A. Cesta, editor, *Proc. 6th European Conf. on Planning*, Toledo, Spain, 2001. Springer: LNCS.

[11] R Fikes and N. Nilsson. STRIPS: a new approach to the application of theorem proving to problem solving. *Artif. Intell.*, 2(3/4):189–208, 1971.

[12] M. Fox and D. Long. The automatic inference of state invariants in TIM. *Journal of AI Research*, 9:367–421, 1998.

[13] H. Geffner. Perspectives on artificial intelligence planning. In *Eighteenth national conference on Artificial intelligence*, pages 1013–1023, Menlo Park, CA, USA, 2002. American Association for Artificial Intelligence.

[14] A. Gerevini and D. Long. Plan constraints and preferences in PDDL3. Technical report, Department of Electronics for Automation, University of Brescia.

[15] A. Gerevini and L. Schubert. Inferring state constraints for Domain-Independent Planning. In *AAAI/IAAI*, pages 905–912, 1998.

[16] P. Haslum. *Admissible Heuristics for Automated Planning*. PhD thesis, Linkping University, Sweden, 2006.

[17] P. Haslum, B. Bonet, and H. Geffner. New admissible heuristics for domain-independent planning. In *AAAI*, pages 1163–1168, 2005.

[18] P. Haslum and H. Geffner. Admissible heuristic for optimal planning. In S. Chien, S. Kambhampati, and C. Knoblock, editors, *Proc. 6th International Conf. on Artificial Intelligence Planning and Scheduling*, pages 140–149, Breckenridge, CO, 2000. AAAI Press.

[19] M. Helmert. A planning heuristic based on causal graph analysis. In *ICAPS*, pages 161–170, 2004.

[20] J. Hoffmann and B. Nebel. Ff domain collection. http://members.deri.at/ joergh/ff-domains.html (01 de marzo de 2007).

[21] J. Hoffmann and B. Nebel. The FF planning system: Fast plan generation through heuristic search. *Journal of Artificial Intelligence Research*, 14:253–302, 2001.

[22] R. Holte, J. Newton, A. Felner, R. Meshulam, and D. Furcy. Multiple pattern databases. In *ICAPS*, pages 122–131, 2004.

[23] H. Kautz and B. Selman. Unifying sat-based and graph-based planning. In *IJCAI '99: Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence*, pages 318–325, San Francisco, CA, USA, 1999. Morgan Kaufmann Publishers Inc.

[24] R. Korf. Depth-first iterative-depeening: An optimal admissible tree search. *Artificial Intelligence*, 27(1):97–109, 1985.

[25] R. Korf. Finding optimal solutions to Rubik's Cube using pattern databases. *Proc. 14th National Conf. on Artificial Intelligence*, pages 700–705, 1997.

[26] R. Korf and Taylor L. Finding optimal solutions to the Twenty-Four Puzzle. In W. Clancey and D. Weld, editors, *Proc. 13th National Conf. on Artificial Intelligence*, pages 1202–1207, Portland, OR, 1996. AAAI Press / MIT Press.

[27] M. Luby. A simple parallel algorithm for the maximal independent set problem. *SIAM J. Comput.*, 15(4):1036–1055, 1986.

[28] D. McDermott. PDDL — the planning domain definition language, 1998. AIPS-98 Competition Committee.

[29] D. McDermott. Using regression-match graphs to control search in planning. *Artificial Intelligence*, 109(1-2):111–159, 1999.

[30] N. Nilsson. *Principles of Artificial Intelligence*. Tioga, Palo Alto, 1980.

[31] J. Pearl. *Heuristics*. Morgan Kaufmann, 1983.

[32] E. Pednault. ADL: exploring the middle ground between STRIPS and the situation calculus. In *Proceedings of the first international conference on Principles of knowledge representation and reasoning*, pages 324–332. Morgan Kaufmann Publishers Inc., 1989.

[33] D. Weld. An introduction to least commitment planning. *Artificial Intelligence Magazine*, 15(4):27–61, 1994.

[34] R. Zhou and E. Hansen. Breadth-first heuristic search. *Artifial Intelligence*, 170(4-5):385–408, 2006.