Técnicas de Optimización para el Proceso de Grounding en Planning Clásico

por

Facundo Jose Bustos

Presentado ante la Facultad de Matemática, Astronomía, Física y Computación como parte de los requerimientos para la obtención del grado de Doctor en Ciencias de la Computación de la

UNIVERSIDAD NACIONAL DE CÓRDOBA

Mayo, 2020

Director: Dr. Carlos Areces
Codirector: Dr. Martin Dominguez

Tribunal Especial:

Dr. Pedro D'Argenio de la Universidad Nacional de Córdoba
Dr. Sebastián Uchitel de la Universidad de Buenos Aires
Dr. Valentin Cassano de la Universidad Nacional de Río Cuarto

Resumen

La planificación automática, o simplemente *Planning*, es una de las áreas más antiguas y centrales dentro de la Inteligencia Artificial. En ella, se estudia la generación automática de acciones por parte de agentes inteligentes con el objetivo de alcanzar una meta a partir de una situación inicial. Los planificadores son algoritmos que computan el comportamiento de los agentes a partir de una descripción detallada de los mismos y su entorno. Actualmente son considerados como *"resolutores automáticos"* (automatic solvers) para cierta clase de modelos matemáticos, que van desde entornos completamente observables (sin incertidumbre) y acciones determinísticas, hasta entornos con observación parcial (incertidumbre) y acciones con efectos estocásticos. En todos estos posibles escenarios, la generación de acciones, desde un modelo es en general un problema intratable. Por lo tanto, el desafío central en el área de planning es el nivel de escalabilidad. Los diferentes métodos de planning, explotan la estructura interna del problema. En particular, los planificadores modernos, trabajan en una representación de bajo nivel de abstracción de un problema de planning. Esta representación es computada a partir de una representación de más alto nivel, mediante un proceso que se conoce como *"proceso de grounding"*, el cual, se realiza previamente a la búsqueda de una solución del problema. El proceso de grounding, ha demostrado ser un obstáculo importante, sobre todo en aquellos problemas, donde la dimensión de la representación de bajo nivel agota los recursos computacionales, ya sea, tiempo y/o memoria disponibles en el planificador. En este trabajo se presentan dos técnicas que pretenden optimizar el resultado del proceso de grounding, que llevan a cabo los planificadores, con el fin de conseguir una representación de bajo nivel más eficiente del problema. Una de ellas opera directamente en la representación de más alto nivel del problema, transformando su estructura, para luego ser sometido al proceso de grounding. La otra técnica, modifica el proceso de grounding propiamente. Por supuesto, ambas técnicas preservan cierta noción de equivalencia con respecto a la representación anterior.

La primer técnica presentada en este trabajo, llamada *"Action Schema*

Splitting", consiste en dividir las acciones de un problema en sub-acciones, tales que, cada una de ellas ejecuta una parte de la acción original. Esta operación no es trivial, ya que, para preservar equivalencia necesitamos introducir varios conceptos relativos al orden en que las acciones son ejecutadas. Además, desde el punto de vista computacional, obtener un conjunto de sub-acciones óptimo a partir de una acción es un problema NP-completo. El reemplazo de acciones por sub-acciones puede ocasionar una importante reducción en el tamaño de la representación de bajo nivel sobre la cual se realiza finalmente la búsqueda heurística para hallar una solución. Esta técnica constituye un compromiso entre la distancia a la meta y el tamaño de la representación de bajo nivel del problema.

La segunda técnica que se presenta en este trabajo, llamada *"Subdominización"*, consiste en modificar directamente el algoritmo que implementa el proceso de grounding del planificador. La idea se basa en generar únicamente aquellas acciones que son realmente necesarias para alcanzar la meta, descartando el resto. Por lo tanto, el éxito de esta técnica depende de la precisión para detectar las acciones más relevantes del problema. Actualmente el proceso de grounding utiliza una versión relajada del problema, para extraer información valiosa sobre las acciones, que luego, es utilizada para resolver el problema original (no relajado). Nuestra técnica consiste en una optimización de este proceso, computando una versión parcial del mismo. Si conseguimos ser lo suficientemente inteligentes para obtener una proyección relevante del problema, esto se traduce en un ahorro significativo en la cantidad de acciones del problema, y en consecuencia, reducimos el tamaño de la representación de bajo nivel del mismo.

Nuestros experimentos demuestran que ambas técnicas pueden constituirse en herramientas prácticas muy útiles dentro del área, porque logran mejorar en muchos casos el resultado del proceso de grounding del planificador. Además, son técnicas valiosas desde un punto de vista teórico, en el sentido de que el proceso de grounding, nunca había sido exhaustivamente tratado o estudiado por la comunidad de planning hasta este momento. Es por eso que consideramos que este trabajo suple esta carencia. En definitiva, pretendemos que este trabajo sea recién un punto de partida para nuevas preguntas e interrogantes sobre cómo abordar el problema de grounding en los planificadores modernos.

Summary

Automatic planning or simply *Planning*, is one of the areas oldest and central within Artificial Intelligence. In it, the automatic generation of actions by intelligent agents with the aim of reaching a goal from an initial situation is studied. Planners are algorithms that compute the behavior of agents from of a detailed description of them and their environment. They are currently considered as *"automatic solvers"* for a certain class of mathematical models, ranging from fully observable environments (no uncertainty) and deterministic actions, to partially observed environments (uncertainty) and actions with stochastic effects. All these possible scenarios, the generation of actions from a model is generally an intractable problem. Therefore, the central challenge in the planning area is the level of scalability. The different planning methods exploit the internal structure of the problem. Modern planners, in particular, work on a low-level representation of a planning problem. This representation is computed from a higher level representation, through a process known as *"grounding process"*, which is carried out prior to searching for a solution to the problem. Thus grounding, has proven to be an important obstacle, especially in those problems where the dimension of the low-level representation exhausts the computational resources (time and/or memory) available in the planner.In this work, two techniques are introduced with aim to optimize the result of the grounding process, which is carried out by the planner, in order to achieve a more efficient low-level representation of the problem. The first technique operates directly on the highest level representation of the problem, transforming its structure, to then be subjected to the grounding process. The second technique modifies the grounding process itself. Of course, both techniques preserve some notion of equivalence regarding to the original representation.

The first technique presented in this work, called *"Action Schema Splitting"* consists to split the actions of a problem into sub-actions, such that each of them execute a part of the original action. This operation is not trivial, since, to preserve equivalence, we need to introduce several concepts related to the order in which the actions are executed. Also, from a computational

point of view, obtaining an optimal set of sub-actions from an action is a NP-completo problem. The replacement of actions by sub-actions can cause a significant reduction in the size of the low-level representation on which the heuristic search is finally performed to find a solution. This technique constitutes a trade-off between the distance to the goal and the size of the low-level representation of the problem.

The second technique presented in this work, called *"Subdominization"*, consists to directly modifying the algorithm that implements the planner's grounding process. The idea is based on generating only those actions that are really necessary to achieve the goal discarding the rest. Therefore, the success of this technique depends on the precision to detect the most relevant actions of the problem. Currently, the grounding process uses a relaxed version of the problem to extract valuable information about the actions, which is then used to solve the original (unrelaxed) problem. Our technique, consists of optimizing this process, computing a partial version of it. If we able to detect a relevant projection of the problem, this translates into a significant saving in the number of actions of the problem, and consequently, we reduce the size of the low-level representation of it.

Our experiments show that both techniques can become very useful practical tools in the area, because in many cases they improve the result of the planner's grounding process. In addition, they are valuable techniques from a theoretical point of view, in the sense that the grounding process had never been exhaustively treated or studied by the planning community until now. That is why we consider that this work fills this gap. In short, we intend that this work is just a starting point for new questions about how to address the grounding problem in modern planners.

Agradecimientos

Al Consejo Nacional de Investigaciones Científicas y Técnicas (CONICET) por haber financiado durante 5 años mis estudios doctorales. A la Facultad de Matemática, Astronomía, Física y Computación (FaMAF) de la Universidad Nacional de Córdoba (UNC) por haberme brindado durante el mismo período las instalaciones e infraestructura necesaria para realizar mi trabajo de investigación. A la Secretaria de Ciencia y Tecnología (SECyT) dependiente del Ministerio de Ciencia y Tecnología de la Nación (MINCyT) por haber financiado dos de las tres estadías de investigación que realice durante mi doctorado en la Universidad de Saarland, Alemania. Al Dr. Carlos Areces y al Dr. Martin Dominguez por haberme guiado como director y codirector durante este proceso, brindándome ambos una parte importante de su tiempo, por su paciencia y su compromiso con el desarrollo de mi doctorado en temáticas de mi interés. A los colaboradores externos pertenecientes al Grupo de Investigación "Foundations of Artificial Intelligence" (FAI), Dr. Jörg Hoffmann, Dr. Alvaro Torralba y Dr. Daniel Gnad por haberme recibido amablemente en su grupo en tres oportunidades y haber colaborado abiertamente con el desarrollo de nuestras ideas brindándome toda su experiencia en el área.

A todos muchas gracias!

Índice general

Capítulo 1

Introducción

La planificación automática, o simplemente *Planning*, es una de las áreas más antiguas y centrales dentro de la Inteligencia Artificial (AI). En ella, se estudia la generación automática de acciones por parte de agentes inteligentes con el objetivo de alcanzar una meta a partir de una situación inicial.

Los planificadores automáticos son algoritmos que computan el comportamiento de un agente en un dominio dado a partir de una descripción detallada de las acciones disponibles y de su entorno. Los planificadores modernos son considerados como *"solucionadores automáticos"* para cierta clase de modelos matemáticos, que van desde entornos completamente observables (sin incertidumbre) y acciones determinísticas hasta entornos con observación parcial (incertidumbre) y acciones con efectos estocásticos. En todos estos posibles escenarios la generación de una secuencia de acciones adecuada en un modelo dado es, en general, un problema intratable. Es decir, no se conocen soluciones generales que requieran sólo un número polinomial de pasos. Por lo tanto, el desafío central en el área de planning es a nivel de escalabilidad. Ante esta realidad, lo que se intenta responder es: ¿se puden definir heurísticas que permitan encontrar soluciones en un tiempo razonable, en un buen número de situaciones?

Los diferentes métodos de planning explotan la estructura interna del problema. En particular, los planificadores modernos trabajan en una representación de bajo nivel de abstracción de un problema de planning. Esta representación es computada a partir de una representación de más alto nivel, mediante un proceso que se conoce como *"proceso de grounding"*. Este proceso se realiza previamente a la búsqueda de una solución del problema. El proceso de grounding ha demostrado ser un obstáculo importante, sobre todo en aquellas problemas donde la dimensión de la representación de bajo nivel agota los recursos computacionales, ya sea, tiempo y/o memoria disponibles en el planificador.

En esta tesis presentaremos dos técnicas para optimizar el resultado del proceso de grounding que llevan a cabo los planificadores, con el fin de conseguir una representación de bajo nivel más eficiente del problema.

Presentaremos primero una técnica llamada *"Action Schema Splitting"*, que opera directamente en la representación de más alto nivel del problema transformando su estructura, para luego, ser sometida al proceso de grounding. La segunda técnica llamada *"Subdominización"* modifica el proceso de grounding propiamente. Por supuesto, deberemos mostrar que ambas técnicas preservan cierta noción de equivalencia con respecto a la representación anterior. Es decir, ninguna de ellas altera el espacio de soluciones (planes) del problema.

En este primer capítulo comenzaremos presentando una breve introducción a Planning, entendida como un área dentro de la Inteligencia Artificial. Presentaremos las primeras ideas subyacentes de nuestras dos técnicas de optimización, finalizando con una descripción sobre cómo está estructurada esta tesis doctoral.

1.1. Planning Clásico

Planning es una disciplina que estudia la derivación automática de comportamientos por parte de agentes inteligentes a partir de una descripción lógica de un entorno, una situación inicial y una meta a alcanzar. El desafío principal en Planning consiste en explotar la estructura interna de los modelos lógicos, permitiendo escalar a instancias donde la búsqueda por fuerza bruta no es viable.

La disciplina de planning combina dos grandes áreas de la IA: métodos de búsqueda y lógica. Un sistema de planning o simplemente un *planificador* es un algoritmo que toma un descripción lógica de un problema de planning y automáticamente computa una solución del mismo. La fertilización cruzada y el intercambio de ideas entre ambas áreas, ha permitido importantes avances en las últimas dos décadas, lo cual, se tradujo en un aumento del uso de planificadores en aplicaciones industriales.

Planning como disciplina no solo emerge de la investigación en áreas tales como búsquedas en espacios de estados, demostración de teoremas y teoría de control, sino también desde necesidades prácticas en robótica, logística y otros dominios. Desafortunadamente, no se tiene una clara comprensión de qué técnicas funcionan mejor en qué clase de problemas. Desde el punto de vista computacional, es un problema difícil. El problema de decidir si un plan existe, aún para lenguajes de descripcion de dominios simples, es PSPACE-completo. Este resultado evidencia la importancia de hallar técni-

cas y algoritmos que permitan encontrar en la práctica soluciones con tiempos de respuesta aceptables.

Actualmente, los planificadores operan sobre una representación proposicional explícita de estados y acciones. Esta representación facilita la obtención de heurísticas de búsqueda efectivas, que derivan en el desarrollo de poderosos y flexibles algoritmos para la resolución de problemas.

Existen diferentes tipos o paradigmas de problemas de planning que están determinados por cuan variable, dinámico, observable y predecible es el entorno que se modela. Algunas de las características del entorno a modelar que determina el paradigma de planning son:

- *La dinámica del entorno.* Un agente puede asumir que se encuentra en un mundo estático y sólo los efectos de sus propias acciones tienen la capacidad de modificar el estado del entorno. O puede asumir que ocurren eventos exógenos, que pueden cambiar el entorno, independientemente de su accionar.

- *Observabilidad del entorno.* Si el agente conoce absolutamente toda la información del entorno, en todo momento, o si el agente puede desconocer cierta información del estado actual.

- *Incertidumbre del agente.* Si el agente es capaz de tratar o no la incertidumbre de su entorno, o de predecir el futuro producto de un entorno no determinístico.

- *Tiempo y Concurrencia.* La ejecución de cada acción puede o no consumir unidades de tiempo. Además, las acciones pueden ser secuenciales, una sola acción se ejecuta a la vez o concurrentes, es decir, con la posiblidad de que más de una acción se esté ejecutando al mismo tiempo.

- *Entorno mono-agente o multi-agente.* Si se trata de un entorno con un único agente inteligente o más de uno. La noción de entornos multiagente, nos lleva a tratar con el concepto de colaboración entre agentes (los agentes se ayudan entre si para alcanzar una meta) o de competencia (los agentes compiten entre ellos para alcanzar la meta).

En particular, cuando suponemos un entorno estático, completamente observable, acciones determísticas, no-estocásticas, no-concurrentes, y monoagente decimos que se trata de un problema de *Planning Clásico*. Claramente, la mayoría de los problemas de la vida real no encuadran con estos supuestos. Entonces, surge la pregunta de por qué estudiamos Planning Clásico. La respuesta es sencilla: debido a sus fuertes supuestos, lo convierten en el paradigma de planning más simple para razonar e implementar ideas. De esta

manera, Planning Clásico se convirtió en el framework más utilizado por la comunidad de planning para realizar investigaciones. Muchas de las ideas del área fueron concebidas en este paradigma y luego extendidas a paradigmas más complejos. Además, se ha demostrado que paradigmas más complejos pueden ser convertidos a un problema clásico equivalente, lo cual, indica que el paradigma clásico tiene un nivel de expresividad similar a muchos de los paradigmas más complejos. Es por eso que, todas las técnicas que se presentan en este libro son siempre diseñadas para el paradigma clásico.

Un problema de planning clásico puede ser descripto en términos de una situación inicial de partida, una meta a alcanzar y ciertas acciones que un agente puede llevar a cabo. Entonces para resolver un problema debemos determinar si existe una secuencia de acciones, tal que, al llevarse a cabo, se alcanza la meta, partiendo de la situación inicial. Tal secuencia de acciones se dice que es un *plan*.

Sobre un mismo problema de planning surgen diferentes preguntas interesantes, cuyas respuestas generan distintas clases de problemas: e. g., existencia de plan (*plan existance*), construcción de algún plan (*satisficing planning*), y construcción del mejor plan (*optimal planning*).

En el problema de existencia de plan dado un problema de planning, se debe decidir si existe o no un plan. En los otros dos problemas, se debe exhibir un plan adecuado. En un caso, cualquier plan alcanza, mientras que en el segundo, se debe encontrar el plan que minimiza alguna noción de costo. A pesar de que se sabe que estos problemas son PSPACE-completos [Bylander, 1994], en la práctica optimal planning, es más difícil que satisfacing planning, es decir, comprobar que un plan es óptimo es, en la mayoría de los casos, más costoso que simplemente encontrar algún plan.

La comunidad de planning ha propuesto diferentes enfoques para resolver el problema de planning, entre los que podemos mencionar:

- *Planning como Satisfabilidad:* una tarea de planning puede ser codificada mediante diferentes métodos en una fórmula booleana, donde, cada asignación que hace a la fórmula satisfacible, se corresponde con un plan de la tarea de planning. Por lo tanto, con cualquier demostrador SAT podemos encontrar un plan para nuestra tarea, vía la asignación devuelta por el demostrador [Kautz and Selman, 1992; Kautz and Selman, 1998]. Este enfoque implica que cualquier avance importante en el estado del arte de los algoritmos de SAT, conlleva directamente, a un avance en la resolución de tareas de planning.

- *Planning como Búsqueda Heurística:* la mayoría de los planificadores emplean algoritmos de búsqueda heurística [Bonet and Geffner, 1998].

Estos algoritmos exploran el espacio de estados asociado a la tarea de planning con el fin de encontrar un camino entre el estado inicial y algún estado que satisface la meta. Para tratar con la naturaleza exponencial del espacio de estados, los algoritmos de búqueda incorporan una función heurística, cuya finalidad es, informar al algoritmo sobre qué parte del espacio de estados resulta más relevante a la hora de hallar un camino hacia la meta, ignorando gran parte del espacio de estados por no ser considerado relevante por la heurística.

- *Planning como Análisis de Grafo:* se basa en el análisis de una estrucura de datos estratificada denominada *Planning Graph* que contiene codificada de una manera conveniente toda la información necesaria para decidir cuestiones de alcanzabilidad y evitar backtracking [Kambhampati *et al.*, 1997].

- *Planning como Orden Parcial:* en este enfoque un plan es visto como un conjunto de acciones y una relación de orden parcial sobre este conjunto que especifica qué acciones deben ser ejecutadas previamente con respecto a otras. La diferencia con un plan secuencial radica en que la relación de orden sólo es impuesta cuando es estrictamente necesaria. Esto proporciona mayor flexibilidad de ejecución. Además, reduce el número de planes, ya que, podemos pensar que un plan parcialmente ordenado denota todos los planes secuenciales que se obtienen al variar el orden de ejecución de las acciones sobre las cuales no hay una restricción de orden impuesta [Penberthy and Weld, 1992; Younes and Simmons, 2003].

Para especificar un problema de planning es necesario tener un lenguaje formal donde se describen todos los detalles del mismo. PDDL (Planning Domain Description Language), es el lenguaje estándar para la definición de problemas de planning. Originalmente introducido en el año 1998 [McDermott *et al.*, 1998], con el objetivo de ser el lenguaje de especificación común para las diferentes ediciones de la competencias internacionales de planning (International Planning Competition, IPC).

PDDL separa la definición de un problema de planning en dos partes: el dominio de planning y la tarea de planning. En el dominio se definen las acciones disponibles, mientras que en la tarea se define el estado inicial y la meta a alcanzar junto con los objetos del entorno a modelar.

PDDL utiliza lógica de predicados, ya que, esto permite definir las acciones en forma más compacta y flexible. Sin embargo, los planificadores trabajan sobre una representación de más bajo nivel que resulta conveniente para la aplicación de técnicas de búsqueda en forma directa. Es por eso que

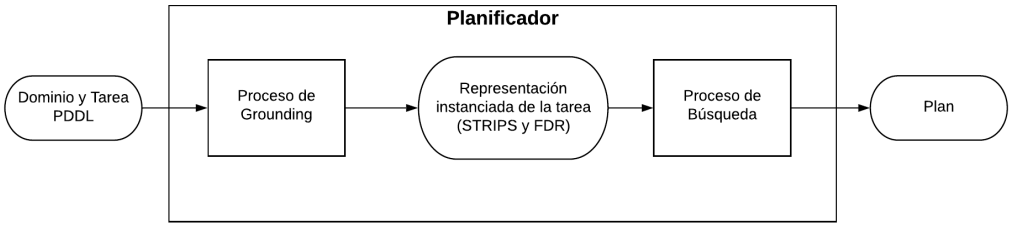


Figura 1.1: Diagrama de Flujo (pipeline) de un planificador basado en búsqueda heurística.

la mayoría de los planificadores transforman la especificación PDDL de un problema de planning a una representación proposicional (llamada formato STRIPS) y/o funcional (llamada formato FDR) [Helmert, 2009], más aptas para realizar la búsqueda de un plan, y que describiremos en detalle más adelante. Esta transformación se conoce como *Proceso de Grounding* y tiene lugar dentro del planificador como una etapa de pre-procesamiento previo a la etapa de búsqueda heurística (ver figura 1.1).

1.2. Sobre Nuestro Trabajo

En esta tesis presentamos dos técnicas innovadoras que apuntan a optimizar el proceso de grounding de un planificador.

El proceso de grounding, es exponencial en la aridad de los esquemas de acción y en los predicados utilizados para modelar un problema. Para un problema dado, el proceso de grounding, es polinomial en la cantidad de objetos definidos del problema. En la práctica, ocurren problemas o tareas de planning donde el proceso de obtención de la representación proposicional del problema agota los recursos computacionales (tiempo y/o memoria) del planificador. Este fenómeno, impide el comienzo de la etapa de búsqueda, por lo tanto, resulta clave obtener métodos que logren a optimizar dicho proceso ofreciendo tiempos de respuesta aceptables.

Para intentar resolver éste problema, presentamos la técnica *Action Schema Splitting*, como una optimización que actúa en la descripción PDDL del problema. La expectativa es que, la transformación efectuada por la técnica, produzca una reducción importante en la cantidad de acciones del problema, y por ende, una disminución en el tamaño de la representación proposicional del mismo. La Figura 1.2, pretende dar una primera idea sobre dónde opera puntualmente esta técnica, dentro del pipeline del planificador. Esta técnica constituye un compromiso entre la reducción de la cantidad de acciones pro-

posicionales y el incremento de la distancia a la meta. Más adelante, veremos que los resultados obtenidos de los experimentos, sobre distintos problemas, demuestran la efectividad de la técnica.

Además, presentamos una segunda técnica de optimización, llamada *Subdominización*, que tiene lugar dentro del propio proceso de grounding del planificador. Esta técnica, consiste en una modificación del algoritmo de grounding, actualmente implementado dentro del planificador Fast-Downward (FD) [Helmert, 2006], por ser éste, el framework de planning más utilizado dentro de la comunidad. Más precisamente, modificamos el orden en que las acciones son generadas por el algoritmo, tratando de priorizar aquellas, que parecen ser más relevantes para resolver el problema, con la "esepranza" de que estas acciones sean suficientes para solucionar el problema, y de esta forma, poder descartar el resto. Al igual que como hicimos con el caso de la técnica de splitting, la figura 1.3 intenta graficar, dónde opera puntualmente la optimización por subdominización, dentro del pipeline del planificador. Veremos que los resultados empíricos evidencian que esta técnica, en mucho problemas, puede significar una alternativa muy efectiva para mejorar el resultado obtenido del proceso de grounding, y así, encontrar planes, para tareas que de otra manera no son tratables.

En general, ambas técnicas han demostrado ser una herramienta útil en aquellos dominios donde la representación proposicional de la tarea no puede ser totalmente generada por el proceso de grounding. Permitiendo ahora en estos casos su generación, y por ende, el comienzo de la etapa de búsqueda. Además, desde un punto de vista de la complejidad computacional, como el proceso de búsqueda enfrenta un problema PSPACE-completo con respecto al tamaño de la tarea proposicional, ambas técnicas consiguen reducir el tamaño de la tarea. En particular, disminuyendo la cantidad de acciones proposicionales. De esta forma, logramos que la etapa de búsqueda se realice sobre una tarea de planning más pequeña. Obviamente que tal recorte de acciones, se realiza de una manera que garantiza que ambas técnicas sean correctas y completas.[1]

1.3. Outline

Esta tesis se organiza en las siguientes cuatro partes:

- Parte I: *Planning Clásico*. Contiene todo lo referido a los conceptos básicos de planning clásico, necesarios para la posterior comprensión

[1] Más adelante veremos que la completitud de la técnica de subdominización estrictamente hablando se alcanza en el límite.

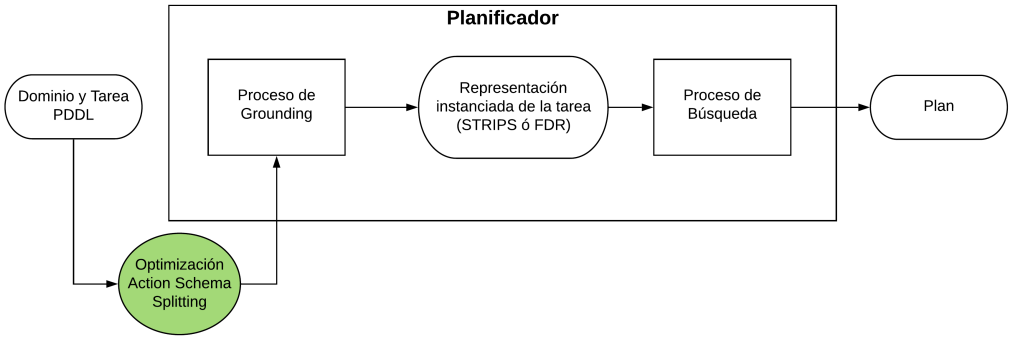


Figura 1.2: Diagrama de Flujo (pipeline) correspondiente a la optimización por Action Schema Splitting.

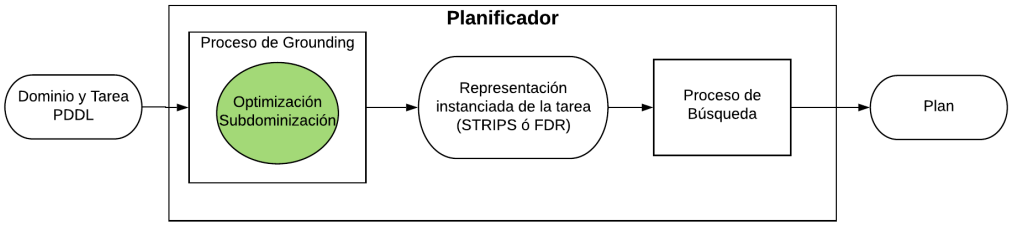


Figura 1.3: Diagrama de Flujo (pipeline) correspondiente a la optimización por Subdominización.

de nuestras técnicas de optimización. En el primer capítulo, definimos el paradigma clásico, introducimos la noción de estados, acciones y meta. Estudiamos la complejidad computacional del problema de planning clásico, visto como un problema de decisión. Luego, planteamos el problema de planning como un problema de búsqueda heurística. Estudiamos la relajación más utilizada en planning, relajación por borrado (delete relaxation), para obtener heurísticas tratables. Por último, presentamos brevemente el lenguaje estándar PDDL utilizado para especificar problemas de planning. En el segundo capítulo, introducimos en detalle, los conceptos relativos al proceso de grounding que realizan los planificadores, ya que, en definitiva es el proceso apuntado por nuestras dos técnicas de optimización.

- Parte II: *Action Schema Splitting*. Contiene todo lo referido a nuestra primer técnica de optimización. En el primer capítulo introducimos los conceptos fundamentales de la técnica de splitting como validez, secuen-

cializaciones correctas, etc. Luego, estudiamos la complejidad computacional de obtener un splitting y presentamos un algoritmo voraz para ello. Después, extendemos la técnica para el caso de acciones ADL, y por último, introducimos la noción de "splitability", que intenta caracterizar aquellos problemas que son más adecuadas para ser optimizados mediante la función de split. En el segundo capítulo de esta parte, se evalúan y analizan los resultados empíricos obtenidos de los diferentes experimentos realizados sobre una amplia variedad de problemas. El último capítulo de esta parte, contiene conclusiones y consideraciones generales sobre la técnica de splitting.

- Parte III: *Subdominización*. Contiene todo lo referido a nuestra segunda técnica de optimización. En el primer capítulo de esta parte, introducimos los conceptos fundamentales del proceso de grounding parcial, tales como: acciones relevantes, condición de parada, el problema de la verificación, estrategia incremental, correctitud y completitud de la técnica. En el segundo capítulo de esta parte, se evalúan y analizan los resultados empíricos obtenidos de los distintos experimentos realizados, con el fin evaluar la efectividad de la técnica y cada uno de sus conceptos. En el último capítulo de esta parte, tenemos las conclusiones generales de la técnica de subdominización.

- Parte IV: *Conclusiones y Futuro*. La cuarta y última parte de este libro contiene nuestras conclusiones y consideraciones generales como producto de todo el trabajo realizado hasta el momento, así como, posibles lineas futuras de investigación.

Parte I

Planning Clásico

Capítulo 2

Conceptos básicos de Planning Clásico

2.1. Introducción

Planning es el approach basado en modelos descriptivos para la derivación de acciones a ejecutar por agentes autónomos inteligentes. Los agentes deben seleccionar acciones, a partir de un modelo que especifica como éstas funcionan, la situación actual y la meta a lograr. El desafío principal en Planning consiste en explotar la estructura interna de los problemas permitiendo escalar a instancias donde la fuerza bruta es inpracticable.

Actualmente *Planning* es un área de gran interés dentro de la Inteligencia Artificial (IA). Una razón de esto es que combina dos áreas clásicas: métodos de búsqueda y lógica. El cruce de ideas desde ambas áreas ha llevado a un avance en la pasada década de varios órdenes de magnitud y un aumento del uso de planificadores en aplicaciones industriales.

Planning como disciplina emerge de la investigación en áreas tales como búsquedas en un espacio de estados, demostración de teoremas y teoría de control. Y desde necesidades prácticas en robótica, organización y otros dominios. Desafortunadamente, no se tiene aún una clara comprensión de qué técnicas funcionan mejor en qué clase de problemas. Planning puede ser visto como un ejercicio de control de la explosión combinatoria, y desde el punto de vista computacional, es una tarea compleja. De hecho, aún sus versiones más simples son PSPACE-completo. Este resultado evidencia la importancia de hallar técnicas y algoritmos que permitan hallar, en la práctica, soluciones con tiempos de respuesta aceptables.

La mayoría de los planificadores modernos operan con representaciones proposicionales explícitas de estados y acciones. Estas representaciones hacen

posible la obtención de heurísticas de búsquedas efectivas y el desarrollo de poderosos y flexibles algoritmos para la resolución de problemas.

2.2. Estados, acciones y metas

Una tarea de planning puede ser descripta en términos de una situación inicial de partida, una meta a alcanzar y ciertas acciones que un agente puede llevar a cabo. Una solución de la tarea consiste en determinar una secuencia de acciones que permita alcanzar la meta partiendo de la situación inicial. A continuación, presentamos los conceptos básicos de planning con los que debemos empezar a familiarizarnos:

Representación de estados. Cada mundo o estado posible de una tarea de planning se representa en términos de un conjunto de símbolos proposicionales lógicos, como p, q, r, etc. Cada símbolo proposicional modela un aspecto del entorno. Por ejemplo, el símbolo proposicional p puede modelar la situación "el agente A_1 se encuentra en la habitación $room_1$". Sobre los estados se asume la hipótesis de mundo cerrado por lo que todas las símbolos proposicionales que no son mencionados en un estado se asumen falsos en ese estado. De esta manera, el conjunto $\{p, q\}$ representa el estado en el que vale p y q, y son falsos todos los demás símbolos proposicionales.

Representación de la meta. Una meta es representada a través de una fórmula proposicional. Cumplir la meta significa alcanzar un estado en el cual la meta se satisface como fórmula proposicional, cuando el estado se considera como una valuación. Por ejemplo, la fórmula $p \wedge q$ representa la meta "vale p y q", la cual es satisfecha en el estado $\{p, q, r\}$, pero no en el estado $\{q, r\}$ porque no vale p.

Representación de acciones. Las acciones son representadas en términos de una precondición y un efecto. La precondición es una fórmula proposicional que representa la condición necesaria para que la acción pueda ser llevada a cabo. Mientras que el efecto es una fórmula que determina los cambios que produce la ejecución de la acción sobre el entorno. Por ejemplo, sea A la siguiente acción:

$$\begin{aligned} &\mathit{Action} : A \\ &\mathit{pre} : p \\ &\mathit{eff} : \neg p \wedge q \end{aligned}$$

La acción A no es aplicable en el estado $\{q\}$, ya que no se cumple su precondición; pero sí en el estado $\{p\}$ transformandoló en el estado $\{q\}$.

Podemos interpretar las acciones como operadores de transformación de estados, en el sentido de que la ejecución de una acción genera que ciertos símbolos proposicionales empiecen a valer, aquellos que ocurren positivamente en el efecto de la acción y otros pueden dejar de valer, aquellos que ocurren negativamente en el efecto de la acción. Desde una perspectiva lógica, el tipo de fórmula que se permite en la precondición y en el efecto de las acciones, así como también el tipo de fórmula que se permite en la meta, determina el tipo de la tarea de planning. En la próxima sección nos concentramos en estudiar los dos tipos básicos de tarea de planning: STRIPS y ADL. Para los cuales nuestras dos técnicas de optimización se encuentran bien definidas.

2.3. Tareas de planning STRIPS y ADL

El tipo de una tarea de planning está dada por la lógica de las fórmulas que ocurren en las acciones y en la meta. En una tarea de planning de tipo STRIPS, las fórmulas que ocurren son fórmulas proposicionales (PL) en 1-CNF. En otras palabras, las fórmulas son conjunciones de literales. Recordar que un literal es cualquier símbolo proposicional o su negación, i. e., un literal es de la forma p ó $\neg p$ con p un símbolo proposicional cualquiera. Y una fórmula $\varphi \in PL$ está en 1-CNF si $\varphi = \bigwedge_i l_i$ con l_i literales.

Definición 1 *Una fórmula STRIPS es una fórmula $\varphi \in PL$ tal que φ está en forma 1-CNF. Una acción a es del tipo STRIPS si su precondición y su efecto son fórmulas STRIPS. Una meta g es del tipo STRIPS si g es una fórmula STRIPS. Una tarea de planning Π es del tipo STRIPS si su meta y todas sus acciones son del tipo STRIPS.*

En la seccion 2.4 vamos a demostrar que decidir si un plan existe para una tarea de planning tipo STRIPS es PSPACE-completo [Bylander, 1994].

En definitiva, podemos describir la sintaxis de las fórmulas que ocurren en las acciones de una tarea de planning tipo STRIPS de la siguiente manera:

$$\begin{array}{rcl} \langle pre \rangle^{STR} & := & l_1 \wedge \cdots \wedge l_n \\ \langle eff \rangle^{STR} & := & l_{n+1} \wedge \cdots \wedge l_{n+m} \end{array}$$

con l_i literales.

El tipo ADL es muy similar al tipo STRIPS pero con la diferencia de que en la precondición y en la meta se permite cualquier fórmula proposicional $\varphi \in PL$. Además, se introduce la noción de efectos condicionales $\varphi \Rightarrow l$: el

efecto l se ejecuta sólo si su condición individual φ se cumple. Diremos que φ es la guarda del efecto condicional y l el efecto.

Definición 2 *Una fórmula ADL es cualquier fórmula $\varphi \in PL$. Un efecto ADL es una conjunción de efectos condicionales $\bigwedge_i \varphi_i \Rightarrow l_i$ tal que φ_i son fórmulas ADL y l_i literales. Una acción a es ADL si su precondición es una fórmula ADL y su efecto es un efecto ADL. Una meta es ADL si es una fórmula ADL. Una tarea de planning es ADL si la meta es ADL y todas sus acciones son ADL.*

Ahora podemos describir la sintaxis de las acciones ADL de la siguiente manera:

$$\begin{array}{lcl} \langle pre \rangle^{ADL} & := & \varphi \text{ con } \varphi \in PL \\ \langle eff \rangle^{ADL} & := & \varphi_1 \Rightarrow l_1 \wedge \cdots \wedge \varphi_m \Rightarrow l_m \text{ con } \varphi_i \in PL \text{ y } l_i \text{ literales} \end{array}$$

Obviamente, si un conjunto de efectos tiene la misma guarda podemos escribir directamente $\varphi \Rightarrow (l_1 \wedge \cdots \wedge l_m)$ para denotar $\varphi \Rightarrow l_1 \wedge \cdots \wedge \varphi \Rightarrow l_m$.

Veamos que STRIPS y ADL tiene la misma expresividad. Para ello, veamos primero que toda tarea STRIPS se puede transformar a una tarea ADL. Notar que si un efecto l no tiene guarda, entonces podemos considerar a este como el efecto condicional $True \Rightarrow l$. Entonces, los efectos de una acción STRIPS pueden ser visto como efectos condicionales con guarda $True$. Por lo tanto, toda acción STRIPS se puede transformar a una acción ADL, trivialmente. Resta ver que toda acción y meta ADL se puede transformar a una acción y meta STRIPS, respectivamente. Primero eliminamos los efectos condicionales. Supongamos que φ es la precondición y $\varphi_1 \Rightarrow l_1$, $\varphi_2 \Rightarrow l_2$ los efectos condicionales de la acción ADL. Entonces generamos una acción por cada combinación posible de las guardas, en este caso, generamos tres acciones de la siguiente manera:

$Action : A_1$	$Action : A_2$	$Action : A_3$
$pre : \varphi \wedge \varphi_1 \wedge \varphi_2$	$pre : \varphi \wedge \varphi_1 \wedge \neg\varphi_2$	$pre : \varphi \wedge \neg\varphi_1 \wedge \varphi_2$
$eff : l_1 \wedge l_2$	$eff : l_1$	$eff : l_2$

El caso anterior, da un método general para eliminar los efectos condicionales. Resta transformar la precondición y la meta a fórmulas proposicionales en 1-CNF. Sabemos que toda formula proposicional $\varphi \in PL$ tiene una fórmula $\varphi' = \varphi'_1 \vee \cdots \vee \varphi'_k$ en forma DNF equivalente. Por lo tanto, tenemos que cada φ'_i es una fórmula en 1-CNF. Luego, generamos una acción A_i por cada cláusula φ'_i, tal que, $pre(A_i) = \varphi'_i$ y cada una con exactamente los mismos efectos.

En definitiva, STRIPS y ADL tiene la misma expresividad. La transformación que presentamos no es polinomial, pero existe una transformación de ADL a STRIPS polinomial eficiente que no presentaremos (ver, e.g., [Gazen and Knoblock, 1997; Koehler and Hoffmann, 2000; Nebel, 2000]). Esto último, garantiza que STRIPS y ADL tienen la misma complejidad, lo cual nos sugiere que STRIPS es lo suficientemente general y poderoso para ser el framework de investigación elegido por la comunidad de planning para desarrollar e implementar las principales ideas del área. En particular, nuestras dos técnicas de optimización fueron concebidas pensado en tareas de planning STRIPS, pero pueden aplicarse también a ADL. En la próxima sección estudiamos en detalle la complejidad computacional de STRIPS y ADL.

2.4. Complejidad

En esta sección vamos a estudiar la complejidad de las tareas de planning de tipo STRIPS, donde tanto las precondiciones y los efectos de las acciones, y la meta son conjunciones de literales. Para esto, definimos formalmente los conceptos de una tarea de planning STRIPS:

Definición 3 *Una tarea de planning de tipo STRIPS es una 4-upla* $\Pi = (F, A, s_0, g)$ *donde*

- F *es un conjunto finito de símbolos proposicionales (hechos).*
- A *es un conjunto finito de acciones STRIPS*
- $s_0 \subseteq F$*, llamado estado inicial*
- g *una meta STRIPS.*

Un estado s *es cualquier subconjunto de* F*. Diremos que un símbolo proposicional* $p \in F$ *vale en un estado* s *sii* $p \in s$*. Dada una fórmula* φ *en 1-CNF, denotaremos con* φ^+ *(respectivamente,* φ^-*) al conjunto de símbolos proposicionales que ocurren positivamente (respectivamente, negativamente) en* φ*. Por ejemplo, si* $\varphi = p \wedge \neg q$*, entonces* $\varphi^+ = \{p\}$ *y* $\varphi^- = \{q\}$*. Notar que* φ *se* satisface *en el estado* s *(*$s \models \varphi$*) sii* $\varphi^+ \subseteq s$ *y* $\varphi^- \cap s = \emptyset$*, con* $\models$ *la semántica natural de PL. Un estado* s *es un estado meta si* $s \models g$*.*

Una acción STRIPS $a \in A$ *es una 2-upla* $a = \langle pre, eff \rangle$ *donde pre y pos son conjunciones de literales. Una acción* a *es aplicable en el estado* s *sii* $s \models pre(a)$*. El estado resultante de ejecutar la acción* a *en el estado* s *se define como* $R(s, a) = (s \setminus pos(a)^-) \cup pos(a)^+$ *siempre que* a *es aplicable en*

s. Notar de la definición de $R(s, a)$ que si un símbolo proposicional p ocurre positivamente y negativamente, entonces en el estado resultante vale p.

Definimos $R(s, a_1 \dots a_n) = R(R(s, a_1 \dots a_{n-1}), a_n)$ siempre que a_i es aplicable en $R(s, a_1 \dots a_{i-1})$. Diremos que $a_1 \dots a_n \in A$ es un plan de Π si $R(s_0, a_1 \dots a_n) \models g$. Diremos que Π tiene solución si tiene al menos un plan.

Con las definiciones anteriores ya estamos en condiciones de estudiar la complejidad de las tareas de planning STRIPS como un problema de decisión, lo cual resulta en un problema difícil desde el punto de vista computacional. En particular, demostraremos que se trata de un problema PSPACE-completo. También mostraremos que es posible definir restricciones sobre el lenguaje STRIPS reducen considerablemente su complejidad.

Definición 4 *Sea PLANSAT el problema de decidir si una tarea de planning STRIPS tiene un plan.*

Teorema 1 *PLANSAT es PSPACE-completo.*

Demo 1 *Veamos que PLANSAT $\in$ PSPACE. Observar que si un problema de planning $\Pi = (F, A, s_0, g)$ tiene solución, entonces tiene una solución de a lo sumo 2^n acciones, donde $n = |F|$. Por lo tanto, podemos dar el siguiente algoritmo no determinista que corre en espacio polinomial para decidir si existe un plan:*

```
1: i := 2^n
2: s := s_0
3: while i > 0 do
4:    seleccionar no determinísticamente una acción a ∈ A
5:    i := i − 1
6:    s := R(s, a)
7:    if s ⊨ g then
8:       return true
9:    end if
10: end while
11: return false
```

Lo anterior prueba que PLANSAT $\in$ NPSPACE, pero por Savitch Theorem sabemos que NPSPACE = PSPACE, por lo tanto, PLANSAT $\in$ PSPACE.

Ahora veamos que PLANSAT es PSPACE-hard. Sea T un problema en PSPACE y $x_1, \dots x_n$ una instancia de T. Sea $M = (Q, \Gamma, \delta)$ la máquina de Turing determinística que computa a T y m la cantidad polinómica de celdas utilizadas por M. Entonces definimos $\pi = (P, A, s_0, g)$ donde:

$$
\begin{array}{lcl}
F & = & \{in(i,x) : i = 1 \ldots m \wedge x \in \Gamma\} \\
 & & \cup \{at(i,q) : i = 1 \ldots m \wedge q \in Q\} \\
 & & \cup \{accept\} \\
s_0 & = & \{at(1,q_0), in(1,x_1), \ldots, in(n,x_n), in(n+1,\Box), \ldots, in(m,\Box)\} \\
g & = & accept
\end{array}
$$

Resta definir el conjunto de acciones. Intuitivamente, queremos definir una acción por cada transición de M. Entonces, para cada $q \in Q, x \in \Gamma$, $i = 1 \ldots m$ tal que $\delta(q,x) = (q', x', t)$ definimos:

Action: $\delta(q,x,i)$
$pre : at(i,q) \wedge in(i,x)$
$pos : \neg at(i,q) \wedge \neg in(i,x) \wedge at(i,x') \wedge at(i',q')$

$$
con\ i' = \begin{cases} i+1 & \text{si } t = R \\ i & \text{si } t = S \\ i-1 & \text{si } t = L \end{cases} \tag{2.1}
$$

Por último, resta definir la acción que verifica la terminación de M, entonces para cada $i = 1 \ldots m$, definimos:

Action: $Halt(i)$
$pre : at(i, q_{halt})$
$pos : accept$

Claramente, la construcción de π es polinomial con respecto a M y es fácil verificar que M termina sii π tiene un plan. Por lo tanto, PLANSAT es PSPACE-completo.

Corolario 1 *Sea ADL-PLANSAT el problema de decidir si una tarea de planning ADL tiene un plan, entonces ADL-PLANSAT es PSPACE-completo.*

Demo 2 *Ya dijimos que existe una transformación polinomial eficiente de ADL a STRIPS y STRIPS es PSPACE-completo, entonces ADL es PSPACE-completo.*

Ahora, surge la pregunta de si existe algún fragmento de STRIPS con menor complejidad que PSPACE-completo. Vamos a mostrar que podemos contestar esta pregunta afirmativamente.

Definición 5 *Sea $PLANSAT_+$ el problema de decidir si un problema de planning STRIPS tal que no ocurren literales negativos en los efectos de las acciones tiene un plan.*

Teorema 2 *$PLANSAT_+$ es NP-completo.*

Demo 3 *Veamos que $PLANSAT_+$ está en NP. Observar que $eff(a)^- = \emptyset$, entonces $s \subseteq R(s,a)$. Luego si $PLANSAT_+$ tiene solución entonces debe tener una solución de longitud a lo sumo $n = |F|$. Entonces, podemos dar el siguiente algoritmo no determinista que corre en tiempo polinomial para decir si hay un plan:*

```
1: i := n
2: s := s_0
3: while i > 0 do
4:     seleccionar no determiníticamente una accion a ∈ A
5:     i := i − 1
6:     s := R(s, a)
7:     if s ⊨ g then
8:         return true
9:     end if
10: end while
11: return false
```

Veamos que $PLANSAT_+$ es NP-hard. Para ello vamos a probar que existe una reducción polinomial de 3-SAT a $PLANSAT_+$. Sea φ una fórmula en 3-CNF con m cláusulas y sea $\{v_1 \dots v_k\}$ el conjunto de variables que ocurren en φ. Entonces definimos $\Pi_+ = (F, A_+, s_0, g)$ donde:

$$\begin{array}{lcl} F & = & \{T(i) : i = 1 \dots k\} \cup \{F(i) : i = 1 \dots k\} \cup \{C(i) : i = 1 \dots m\} \\ s_0 & = & \emptyset \\ g & = & C_1 \wedge \dots \wedge C_m. \end{array}$$

Resta definir el conjunto de acciones. Para ello, la idea es tener una acción que le asigne un valor de verdad a cada variable si ésta aún no fue ya asignada. Entonces, para $i = 1 \dots k$, definimos:

Action: $AssignTrue(i)$	*Action: $AssignFalse(i)$*
$pre : \neg F(i)$	$pre : \neg T(i)$
$pos : T(i)$	$pos : F(i)$

Además, necesitamos definir una acción por cada forma en la que se puede satisfacer una cláusula. Para ello, definimos para cada cláusula C_j y variable v_i que ocurre positivamente en C_j:

Action: $SatClause(j, i)$
$pre : T(i)$
$pos : C(j)$

Y simétricamnte, para cada cláusula C_j y variable v_i que ocurre negativamente en C_j, definimos:

Action: $SatClause(j, i)$
$pre : F(i)$
$pos : C(j)$

Notar que todas la acciones definidas tienen solo efectos positivas. Observar que la construcción de Π_+ es polinomial con respecto a φ. Finalmente, es fácil verificar que φ es satisfacible sii π_+ tiene un plan. Por lo tanto, $PLANSAT_+$ es NP-completo.

Encontramos, entonces, un fragmento de STRIPS con una complejidad menor al caso general (PSPACE-completo) pero aún intratable (NP-completo). Existirá algún fragmento tratable?

Observar que PLANSAT_+ se puede pensar como una relajación del problema original PLANSAT, en el sentido de que descartamos una parte del problema original (los efectos negativos de las acciones). Tal vez podemos seguir relajando el problema para llegar a un fragmento tratable.

Definición 6 *Sea $PLANSAT_+^+$ el problema de decidir si un problema de planning STRIPS tal que no ocurren literales negativos tiene un plan.*

Teorema 3 *$PLANSAT_+^+ \in P$.*

Demo 4 *Observar que $s \models pre(a)^+$ y $s \models pre(a')^+$, entonces $R(s, a) \models pre(a')^+$. Por lo tanto, sólo necesitamos aplicar acciones arbitrariamente hasta alcanzar la meta o hasta no haber acciones que agreguen nuevos efectos positivos al estado actual lo cual significa que la meta es inalcanzable. Claramente, este procedimiento es polinomial en la cantidad de acciones del problema:*

```
1: s := s_0
2: while seleccionar a ∈ A tal que pre(a)^+ ⊆ s y pos(a)^+ ⊈ s do
3:     s := R(s, a)
4:     if s ⊨ g then
5:         return true
6:     end if
7: end while
8: return false
```

El resultado de complejidad anterior nos sugiere que dado una tarea de planning STRIPS Π, podemos construir un tarea de planning Π_+^+, eliminando

todas las ocurrencias de literales negativos de Π. Y luego, extraer información útil a partir de un plan de Π^+_+, aprovechando que esto es lineal en la cantidad de acciones de la tarea, para computar un plan de Π. Este resultado es el fundamento teórico subyacente de la técnica de relajación que vamos estudiar en detalle en la sección 2.8, ya que, un fragmento de STRIPS puede ser interpretado como una relajación de la tarea original en el siguiente sentido: una relajación consiste simplemente en ignorar parte del problema original. En este caso particular, resulta de ignorar los literales negativos.

Por otra lado, del resultado de complejidad general de STRIPS (PSPACE-completo) se desprende la envergadura del problema computacional que enfrentamos. Una posible interpretación de dicho resultado es que una tarea de planning es lo suficientemente complicado como para no ser computable en tiempo polinomial aún con acceso a no-determinismo. Pero, al mismo tiempo, no es extremadamente complicado porque sí es computable en espacio polinomial.

Recordar que un problema es PSPACE-completo con respecto al tamaño de su input. Esto deja en evidencia la importancia de optimizar el tamaño de dicho input. En nuestro caso en particular, es el tamaño de la tarea de planning. Precisamente nuestras dos técnicas de optimización apuntan a disminuir considerablemente el tamaño de este input. Más formalmente: dada una tarea de planning STRIPS $\Pi = (F, A, s_0, g)$ ambas técnicas construyen una tarea de planning $\Pi' = (F', A', s'_0, g')$, tal que, el tamaño en bits de Π' es menor que Π. En particular, esto se logra reduciendo considerablemente la cantidad de acciones de la tarea original, i. e., $|A'| < |A|$. Asegurando que tal recorte de acciones preserve alguna relación entre ambas tareas, de forma tal de poder recuperar una solución de Π a través de una solución de Π'.

2.5. Fórmulas Libres de Negación

Se dice que una tarea de planning STRIPS es libre de negaciones si en las precondiciones de las acciones y en la meta no ocurren literales negativos. Es fácil chequear que existe una manera de convertir cualquier tarea de planning STRIPS en una versión equivalente libre de negaciones en forma lineal. Básicamente, la idea de la conversión consiste en:

1. Para cada símbolo proposicional p que ocurre negativamente en una precondición o en la meta, crear un símbolo proposicional nuevo $\overline{p}$ y agregarlo al estado inicial.

2. En las precondiciones y en la meta reemplazar las ocurrencias negativas de p por $\overline{p}$.

3. En los efectos de la acciones reemplazar las ocurrencia positivas de p por $p \wedge \neg\overline{p}$ y reemplazar las ocurrencias negativas de p por $\neg p \wedge \overline{p}$.

Este resultado indica que en STRIPS permitir o no la ocurrencia de negaciones de símbolos proposicionales en las precondiciones y en la meta no tiene mayor importancia. Por lo tanto, de aquí en adelante, cuando hagamos referencia a una tarea de tipo STRIPS, implícitamente vamos a estar haciendo referencia a una tarea de tipo STRIPS libre de negaciones. Es importante mencionar que este método de conversión se lleva a cabo dentro del proceso de grounding del planificador que se estudia con más detalle en el capítulo 3.5.

2.6. Representaciones: STRIPS vs. FDR

Vamos a representar una tarea de planning STRIPS de una forma más conveniente utilizando conjuntos de símbolos proposicionales, evitando de esta manera lidiar con fórmulas proposicionales. Una meta o precondición STRIPS es de la forma $p_1 \wedge \ldots p_n$, con p_i símbolos proposicionales y puede ser presentada por el conjunto $\{p_1, \ldots, p_n\}$. Mientras que un efecto STRIPS es de la forma $l_1 \wedge \ldots l_n$ con l_i literales y se puede representar a través de dos conjuntos: uno con los símbolos proposicionales que ocurren positivamente en el efecto y otro conjunto con los símbolos proposicionales que ocurren negativamente en el efecto. Por ejemplo, reconsideremos la acción A definida en la sección 2.2:

$$\begin{aligned} &\mathit{Action} : A \\ &\mathit{pre} : p \\ &\mathit{eff} : \neg p \wedge q \end{aligned}$$

Claramente A es una acción STRIPS. Por lo tanto, podemos representarla por medio de los siguiente tres conjuntos de símbolos proposicionales:

$$\begin{aligned} &\mathit{Action} : A \\ &\mathit{pre} : \{p\} \\ &\mathit{del} : \{p\} \\ &\mathit{add} : \{q\} \end{aligned}$$

La interpretación de esta nueva representación se basa en que la precondición *pre* son los símbolos proposicionales que deben valer para que la acción sea aplicable. Mientras que los conjuntos *add* y *del* representan el conjunto de símbolos proposicionales que agrega y borra la acción producto de su ejecución. Esta representación a través de conjuntos de símbolos proposicionales

nos permite abstraer fórmulas y operadores proposicionales ($\wedge$, $\neg$), lo cual resulta ventajoso para introducir en forma más natural muchas de las técnicas del área. Esta representación también se conoce con el nombre de representación STRIPS, por lo tanto, cuando hablemos de una tarea de planning STRIPS podemos pensar directamente en su representación STRIPS.

Definición 7 (Tarea de Planning STRIPS) *Una tarea de planning STRIPS es una 5-upla* $\Pi = (F, A, c, s_0, G)$ *donde:*

- F *es un conjunto finito de símbolos proposicionales (hechos).*
- A *es un conjunto de finito de acciones STRIPS.*
- $c : A \to \mathbb{R}_0^+$ *es una función de costo.*
- $s_0 \subseteq P$*, llamado estado inicial.*
- $G \subseteq P$*, llamado meta.*

Un estado s es cualquier subconjunto de F. Diremos que un símbolo proposicional $p \in F$ vale en un estado s sii $p \in s$. Una acción STRIPS es una 3-upla $a = (pre, del, add)$, tal que, pre, del y add son subconjuntos de F llamados precondición, lista del y lista add, respectivamente. Denotaremos con $pre(a)$, $del(a)$ y $add(a)$ a la precondición, lista del y lista add de a, respectivamente.

Diremos que a es aplicable en el estado s si $pre(a) \subseteq s$ y definimos el estado resultante de la aplicación de a en el estado s como $R(s, a) = (s \setminus del(a)) \cup add(a)$ siempre que a sea aplicable en s. En forma natural, se define $R(s, a_1 \dots a_n) = R(R(s, a_1 \dots a_{n-1}), a_n)$ siempre que a_i es aplicable en $R(s, a_1 \dots a_{i-1})$. Diremos que $a_1 \dots a_n \in A$ es un plan de Π sii $G \subseteq R(s_0, a_1 \dots a_n)$ y denotamos con $Pl(\Pi)$ al conjunto de todos los planes de Π. Diremos que Π tiene costo uniforme si $c(a) = 1$ para todo $a \in A$.

Notar de la definición de $R(s, a)$ que si un símbolo proposicional p ocurre en la lista *add* y *del* de la acción, entonces en el estado resultante vale p.

Para ejemplificar la definiciones anteriores, proponemos codificar una instancia del problema "Travel Saleman Problem" (TSP) de la Figura2.2 como una tarea de planning STRIPS. Sea

$$\begin{aligned} C \;=\; & \{Sydney(Sy), Adelaide(Ad), Brisbane(Br), \\ & Perth(Pe), Darwin(Da)\} \end{aligned}$$

el conjunto de ciudades a visitar por el viajante. Entonces codificamos TSP como $\Pi^{TSP} = (F, A, c, s_0, g)$ donde:

$P = \{at(x) : x \in C\} \cup \{visited(x) : x \in C\}.$

$s_0 = \{at(Sy), visited(Sy)\}$

$G = at(Sy)\} \wedge \bigwedge_{x \in C} visited(x)$

Luego, para todo x, $y \in C$ tal que existe una camino de x a y (ver Figura 2.2), definimos:

$$\begin{array}{l} Action : Drive(x, y) \\ pre : \{at(x)\} \\ del : \{at(x)\} \\ add : \{at(y), visited(y)\} \end{array}$$

Por lo tanto, tenemos que A = {Drive(Sy, Br), Drive(Br, Sy), Drive(Sy, Ad), Drive(Ad, Sy), Drive(Ad, Da), Drive(Da, Ad), Drive(Ad, Pe), Drive(Pe, Ad)}. Y cada una de ellas con el siguiente costo:

$$c(Drive(x, y)) = \begin{cases} 1 & \text{si } \{x, y\} = \{Sy, Br\} \\ 1.5 & \text{si } \{x, y\} = \{Sy, Ad\} \\ 3.5 & \text{si } \{x, y\} = \{Ad, Pe\} \\ 4 & \text{si } \{x, y\} = \{Ad, Da\} \end{cases} \tag{2.2}$$

Se puede verificar fácilmente que la siguiente secuencia de ocho acciones de A es un plan de Π^{TSP} con un costo total de 20:

1- Drive(Sy, Br)	5- Drive(Da, Ad)
2- Drive(Br, Sy)	6- Drive(Ad, Pe)
3- Drive(Sy, Ad)	7- Drive(Pe, Ad)
4- Drive(Ad,Da)	8- Drive(Ad, Sy)

Una tarea de planning STRIPS define un espacio estados, el cual lo vamos a formalizar mediante un sistema de transición etiquetado, es decir, el espacio de estados de una tarea de planning STRIPS es igual a un sistema de transición etiquetado. Para esto, primero tenemos que introducir el concepto de Sistema de Transición etiquetado (LTS en inglés).

Definición 8 (Sistema de Transición Etiquetado) *Un sistema de transición etiquetado es una 5-upla $\Theta = (S, L, T, I, F)$ donde:*

- *S es un conjunto finito de estados.*
- *L es un conjunto finito de etiquetas (labels) de transición.*
- *$c : L \to \mathbb{R}_0^+$ es una función de costo.*

Figura 2.1: Una instancia del problema TSP.

- $T \subseteq S \times L \times S$ *es una relación de transición.*
- $I \in S$ *es el estado inicial.*
- $F \subseteq S$ *es un conjunto de estados finales.*

El tamaño de Θ *es su cantidad de estados, i. e.* $size(\Theta) = |S|$ *y denotaremos con* $s \in \Theta$ *cuando* $s \in S$*. Decimos que la transición* $(s, l, s') \in \Theta$ *si* $(s, l, s') \in T$ *y escribimos* $s \xrightarrow{l} s'$ *ó* $s \rightarrow s'$ *(cuando la etiqueta no es relevante).*

Θ *es determinístico si para todo* $s \in S$ *y* $l \in L$*, hay a lo sumo un* $s' \in S$*, tal que,* $s \xrightarrow{l} s'$*. Decimos que* Θ *tiene costo uniforme si* $c(l) = 1$ *para todo* $l \in L$*. Un estado* s' *es sucesor de un estado* s *(y* s *es predecesor de* s'*) si* $s \rightarrow s'$*.* s' *es* alcanzable *desde* s *si existe una secuencia de transiciones* $s = s_0 \xrightarrow{l_1} s_1 \ldots s_{n-1} \xrightarrow{l_n} s_n = s'$ *y tanto* $l_1, \ldots, l_n$ *como* $s_0, \ldots, s_n$ *es llamado* path *de* s *a* s'*.* s *es alcanzable si es alcanzable desde el estado inicial* I*. Un estado* s *es* soluble *si existe un path de* s *a* s' *para algún* $s' \in F$*. Una solución de* Θ *es un path del estado inicial* I *a un estado* $s \in F$*. Diremos que* Θ *es soluble si tiene al menos una solución. Y denotamos con* $Sol(\Theta)$ *al conjunto de todas las soluciones de* Θ*.*

Definición 9 (Espacio de Estados STRIPS) *Sea* $\Pi = (F, A, c, s_0, G)$ *una tarea de planning STRIPS, entonces el espacio de estados de* Π *es el sistema de transición etiquetado* $\Theta_\Pi = (S, L, c, T, I, F)$ *donde:*

Figura 2.2: Una instancia más simple del problema TSP.

- $S = 2^F$
- $L = A$
- $T = \{s \xrightarrow{a} s' : pre(a) \subseteq s \ y \ s' = R(s, a)\}$,
- $I = s_0$
- $F = \{s \in S : g \subseteq s\}$.

Con esta definición tenemos que $Pl(\Pi) = Sol(\Theta_\Pi)$. *Y por lo tanto, diremos que* Π *es soluble si* Θ_π *es soluble.*

Veamos el espacio de estados para la codificación STRIPS del problema TSP pero considerando una instancia más simple (ver figura 2.2) con tres ciudades a visitar $C = \{Sydney(Sy), Brisbane(Br), Adelaide(Ad)\}$. En la figura 2.3 se muestra parte del espacio de estados de la versión simplificada de TSP, ya que, sólo se muestran aquellos estados alcanzables desde el estado inicial. Por consiguiente, estados como $\{visited(Sy)\}$ y $\{at(Sy), at(Br)\}$ que pertenecen al espacio de estados no se muestran en la figura por no ser estados alcanzables.[1]

Otra representación muy utilizada dentro del área debido a sus ventajas prácticas consiste en modelar los estados y acciones de una tarea de planning mediante mapeos totales y/o parciales entre variables y sus posibles

[1] $v(x)$ es una abreviación del símbolo proposicional $visited(x)$ para todo $x \in C$.

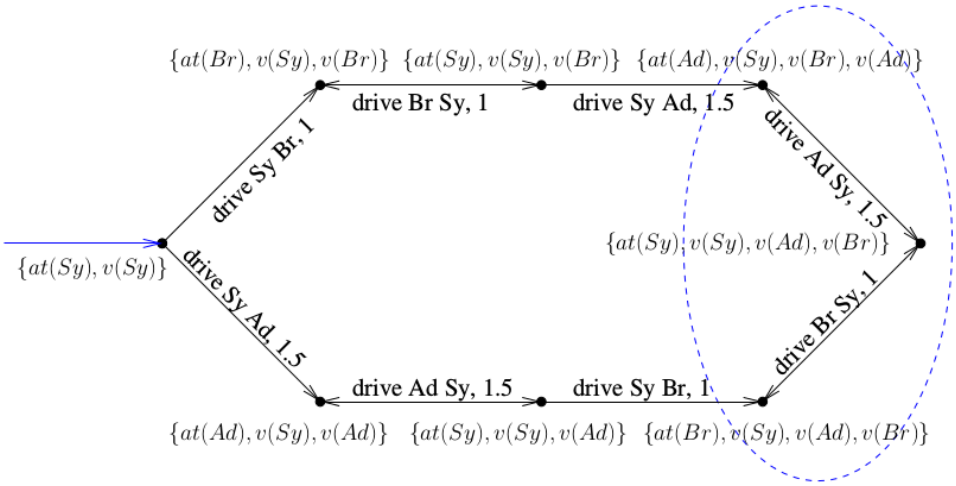


Figura 2.3: Espacio de estados alcanzables de la representación STRIPS del problema TSP simplificado.

valores de asignación. Esta representación conocida como FDR (Finite Domain Representation) o SAS$^+$ propone modelar la tarea en forma funcional [Bäckström, 1992; Bäckström and Nebel, 1993; Bäckström and Nebel, 1995], y no en términos de símbolos proposicionales como es el caso de la representación STRIPS. Por ejemplo, en el problema TSP podemos definir una variable que modela la ubicación actual del viajante y sus posibles valores de asignación son las distintas ciudades a visitar. Esta representación funcional hace explícita el invariante de que el viajante no puede estar en más de una ciudad al mismo tiempo, cosa que no ocurre mediante la representación STRIPS, logrando que el espacio de estados definido por la representación funcional sea más compacto.

Además, es posible transformar una tarea proposicional STRIPS en una tarea funcional FDR y viceversa, en forma muy simple, lo cual demuestra que ambas representaciones tienen el mismo poder de expresividad. Si bien nuestro lenguaje estándar de especificación es PDDL, y este se corresponde más naturalmente con la representación proposicional STRIPS, la mayoría de las herramientas de planning desarrolladas en los últimos años fueron implementadas sobre la plataforma Fast-Downward(FD) [Helmert, 2006], la cual lee una descripción PDDL del problema, obtiene una representación proposicional STRIPS y luego utiliza una transformación "STRIPS-to-FDR"para conseguir una representación funcional compacta y práctica sobre la cual finalmente se realiza la búsqueda heurística para hallar un plan.

Definición 10 (Tarea de Planning FDR) *Una tarea de planning FDR*

es una 5-upla $\Pi = (V, A, c, s_0, G)$ *donde:*

- V *es un conjunto finito de variables, tal que, cada* $v \in V$ *tiene un conjunto finito* D_v *de posibles valores de asignación (dominio).*
- A *es un conjunto finito de acciones FDR.*
- $c : A \to \mathbb{R}_0^+$ *es una función de costo.*
- s_0 *es una asignación total sobre las variables de* V*, llamado estado inicial.*
- G *es una asignación parcial sobre variables de* V*, llamada meta.*

Un estado s *es una asignación total sobre las variables de* V*. Una acción FDR es una 3-upla* $a = (pre, eff)$*, tal que, pre y eff son asignaciones parciales sobre variables de* V *llamadas precondición y efecto, respectivamente. Denotaremos con* $pre(a)$ *y* $eff(a)$ *a la precondición y efecto de* a*, respectivamente. Diremos que* a *es aplicable en el estado* s *si* $pre(a)(v) = s(v)$ *para todo* $v \in Dom(pre(a))$ *y definimos el estado resultante de la aplicación de* a *en el estado* s *como* $R(s, a) = s'$ *donde* $s'(v) = eff(a)(v)$ *si* $v \in Dom(eff(a))$ *y* $s'(v) = s(v)$ *caso contrario, siempre que* a *sea aplicable en* s*. En forma natural, se define* $R(s, a_1 \dots a_n) = R(R(s, a_1 \dots a_{n-1}), a_n)$ *siempre que* a_i *es aplicable en* $R(s, a_1 \dots a_{i-1})$*. Diremos que* $a_1 \dots a_n \in A$ *es un plan de* Π *sii* $G(v) = R(s_0, a_1 \dots a_n)(v)$ *para toda* $v \in Dom(G)$ *y denotamos con* $Pl(\Pi)$ *al conjunto de todos los planes de* Π*. Diremos que* Π *tiene costo uniforme si* $c(a) = 1$ *para todo* $a \in A$*.*

En la representación FDR una acción sobre-escribe los valores asignados a las variables tal como lo indica su efecto, en lugar de agregar y borrar símbolos proposicionales como es el caso de la representación STRIPS. Para ejemplificar las definiciones anteriores, reconsideremos el problema TSP (ver figura 2.2) y codifiquémoslo bajo la representación FDR. Sea $\Pi^{TSP} = (V, A, c, s_0, G)$ donde:

$V = \{at\} \cup \{visited(x) : x \in C\}$ con $D_{at} = C$ y $D_{visited(x)} = \{T, F\}$.

$s_0 = \{at = Sy, visited(Sy) = T, visited(x) = F\}$ con $x \neq Sy$.

$g = \{at = Sy, visited(x) = T\}$ para todo $x \in C$.

Nuevamente para todo x, $y \in C$ tal que existe una camino de x a y (ver Figura 2.2) definimos las siguientes acciones FDR[2]:

[2]La definicion de la función costo c es exactamente la misma que en el caso STRIPS.

$$\begin{aligned} &Action : Drive(x, y) \\ &pre : \{at = x\} \\ &eff : \{at = y, visited(y) = T\} \end{aligned}$$

Dada la naturaleza funcional de la representación FDR, su espacio de estados resulta ser más compacto con respecto al caso STRIPS.

Definición 11 (Espacio de Estados FDR) *Sea* $\Pi = (V, A, c, s_0, g)$ *una tarea de planning FDR, entonces el espacio de estados de* Π *es el sistema de transición etiquetado* $\Theta_\Pi = (S, L, c, T, I, F)$ *donde:*

- S *es el conjunto de asignaciones totales sobre* V
- $L = A$
- $T = \{s \xrightarrow{a} s' : a \text{ es aplicable en } s \text{ y } s' = R(s, a)\}$,
- $I = s_0$
- $F = \{s \in S : s(v) = g(v) \text{ para todo } v \in Dom(g)\}$.

Por lo tanto, tenemos que $Pl(\Pi) = Sol(\Theta_\Pi)$. *Y diremos que* Π *es soluble si* Θ_π *es soluble.*

Reconsideremos la versión simplificada de TSP con sólo tres ciudades $C = \{Sydney(Sy), Brisbane(Br), Adelaide(Ad)\}$. En la figura 2.4 se muestra parte del espacio de estados FDR de la versión simplificada de TSP correspondiente a los estados alcanzables desde el estado inicial. Lo interesante de la figura es notar que aquellos estados que representan la situación donde el viajante se encuentra en más de una ciudad al mismo tiempo e. g., $(\{at(Sy), at(Br)\})$ que si pertenecen al espacio de estados STRIPS, ahora no pertenecen al espacio de estados FDR debido a que la representación funcional no lo permite, dejando en evidencia que el espacio de estados FDR es más compacto.

Por otro lado, ambas representaciones tienen el mismo nivel de expresividad, ya que una representación puede ser transformada en otra.

Para el caso de la transformación "FDR-to-STRIPS" la idea consiste en que para cada variable $v \in V$ con dominio $D_v = \{d_1, \ldots, d_k\}$, tomar k símbolos proposicionales "$v = d_1$", ..., "$v = d_k$". Mientras que para la transformación "STRIPS-to-FDR" la idea es para cada símbolo proposicional $p \in F$, tomar una variable v_p con $D_{v_p} = \{T, F\}$. Nuestras dos técnicas de optimización son más fáciles de introducir bajo la representación STRIPS. De todos modos, al introducir una técnica T para la representación STRIPS,

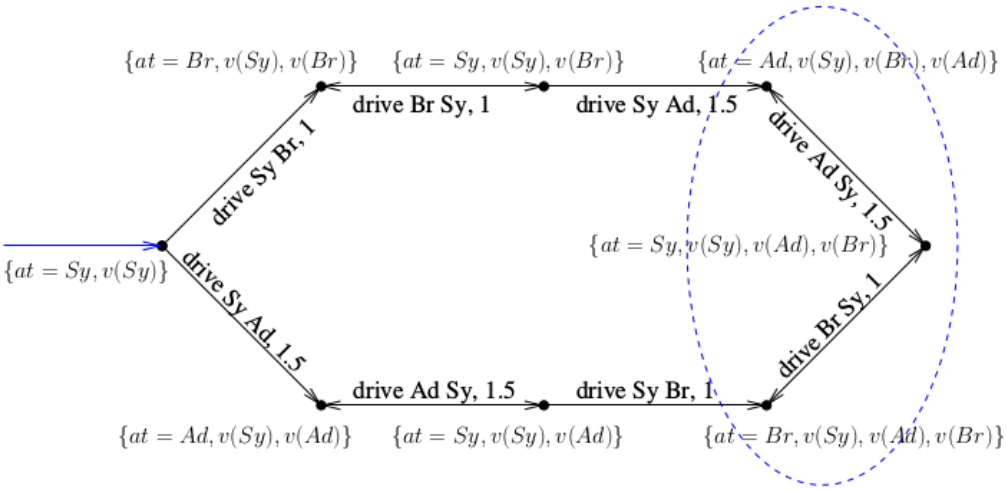


Figura 2.4: Espacio de estados alcanzables de la representación FDR del problema TSP simplificado.

entonces T también es aplicable para la representación FDR haciendo "FDR-to-STRIPS" y luego aplicar T.

Describamos con mayor precisión estas dos transformaciones, teniendo en cuenta que en FDR tenemos mapeos variable-valor, mientras que en STRIPS tenemos símbolos proposicionales.

Definición 12 (FDR-to-STRIPS) *Sea* $\Pi = (V, A, c, s_0, G)$ *una tarea de planning FDR, entonces su conversión STRIPS es la tarea de planning* $\Pi^{STR} = (F, A^{STR}, c, s_0^{STR}, G^{STR})$ *tal que:*

- $F = \{v = d \mid v \in V, d \in D_v\}$ *es el conjunto de símbolos proposicionales.*
- $A^{STR} = \{a^{STR} | a \in A\}$ *donde* $pre(a^{STR}) = \{v = d \mid pre(a)(v) = d\}$, $add(a^{STR}) = \{v = d \mid eff(a)(v) = d\}$ *y* $del(a^{STR}) = \{v = d' \mid eff(a)(v) = d \text{ y } d' \in D_v \setminus \{d\}\}$.
- $s_0^{STR} = \{v = d \mid s_0(v) = d\}$
- $G^{STR} = \{v = d \mid G(v) = d\}$.

La lista *add* establece los nuevos valores de las variables según indica el efecto de la acción, mientras que la lista *del* asegura que cualquier otro valor distinto del que actualmente asigna el efecto, es eliminado.

Proposición 1 *Sea* $\Pi = (V, A, c, s_0, G)$ *una tarea de planning FDR y* Π^{STR} *su conversión STRIPS. Entonces* Θ_Π *y el espacio de estados* $\Theta_{\Pi^{STR}}$ *restringido al conjunto de estados* $s \subseteq F$ *tal que para cada* $v \in V$*, el estado* s *contiene exactamente un símbolo proposicional de la forma* $v = d$ *son isomorfos.*

Definición 13 (Naive STRIPS-to-FDR) *Sea* $\Pi = (F, A, c, s_0, G)$ *una tarea de planning STRIPS, entonces su conversión FDR (naive) es la tarea de planning* $\Pi^{FDR} = (V, A^{FDR}, c, s_0^{FDR}, G^{FDR})$ *tal que:*

- $V = \{v_p \mid p \in F\}$ *el conjunto de variables y* $D_{v_p} = \{T, F\}$.
- $A^{FDR} = \{a^{FDR} | a \in A\}$ *donde* $pre(a^{FDR}) = \{v_p = T \mid p \in pre(a)\}$, $eff(a^{STR}) = \{v_p = T \mid p \in add(a)\} \cup \{v_p = F \mid p \in del(a)\}$
- $s_0^{FDR} = \{v_p = T \mid p \in s_0\}$
- $G^{FDR} = \{v_p = T \mid p \in g\}$

Proposición 2 *Sea* $\Pi = (F, A, c, s_0, g)$ *una tarea de planning STRIPS y* Π^{FDR} *su conversión FDR (naive). Entonces* Θ_Π *y* $\Theta_{\Pi^{FDR}}$ *son isomorfos.*

En esta última transformación todas las variables tiene sólo dos posibles valores, es decir, son todas variables booleanas. Por lo tanto, esta transformación no aporta ninguno de los beneficios de la representación funcional de FDR ya mencionadas. De esto se sigue el nombre de transformación naive.

Surge el interrogante si podemos obtener una transformación STRIPS-to-FDR menos naive que utilice las ventajas de la naturaleza funcional de FDR. La idea consiste en hallar un conjunto de símbolos proposicionales $\{p_1, \ldots, p_k\}$ tal que para cada estado alcanzable s sólo uno de ellos vale, es decir, son símbolos proposicionales mutuamente excluyentes (mutex). Luego para cada conjunto mutex $\{p_1, \ldots, p_k\}$ hallado definir una variable FDR v con $D_v = \{d_1, \ldots, d_k\}$. Esta idea da origen a la transformación STRIPS-to-FDR (clever) [Helmert, 2009]. Claramente, la versión naive es en escencia STRIPS, mientras que la versión clever es más natural y aprovecha mejor la naturaleza funcional de FDR.

Desafortunadamente, decidir si un conjunto de símbolos proposicionales es mutex es PSPACE-completo. Sin embargo, existen algoritmos eficientes que pueden encontrar sólo algunos de estos conjuntos mutex y de esta forma conseguir una mejor representación FDR de la tarea de planning con respecto a la obtenida por la transformación naive. En particular, esta transformación clever es la implementada en el proceso de grounding de la plataforma FD [Helmert, 2006]. Y este es el sistema elegido por nosotros sobre el cual vamos a correr nuestros experimentos cuando evaluemos el efecto resultante de nuestras dos técnicas de optimización.

2.7. Espacios de Búsqueda y Heurísticas

Una vez que tenemos definido la representación de la tarea de planning y por consiguiente su espacio de estados, debemos encontrar una solución. Para ello se propone realizar una búsqueda en el espacio de estados de la tarea de planning. Un *espacio de búsqueda* es un sub-espacio del espacio de estados determinado por un estado de arranque, una función target que identifica un estado como estado de llegada, y una función sucesor definida para cada estado. Más precisamente, un espacio de búsqueda está definido a través de la siguientes tres operaciones:

- $start()$: genera el estado de arranque de la búsqueda.
- $is\text{-}target(s)$: determina si el estado s es un estado de parada de la búsqueda.
- $succ(s)$: genera estados sucesores de s.

Una búsqueda en el espacio de estados puede ser progresiva (hacia adelante), partiendo desde el estado inicial de espacio de estados e intentar llegar a un estado final. O alternativamente, regresiva (hacia atrás), partiendo desde un estado final e intentar llegar al estado inicial. Entonces, la progresividad o regresividad de una búsqueda depende de cómo son instanciadas las tres operaciones anteriores.

Definición 14 (Progresión) *Sea* $\Pi = (F, A, c, s_0, g)$ *una tarea de planning STRIPS, el espacio de búsqueda por progresión de* Π *está dado por:*

- $start() = s_0$
- $is\text{-}target(s) = g \subseteq s$
- $succ(s) = \{(a, s') : s \xrightarrow{a} s' \in \Theta_\pi\}$

En planning, una búsqueda progresiva comienza desde el estado inicial de la tarea y aplica acciones hasta alcanzar un estado meta. De esta manera, explora estados que son alcanzables pero también estados no solubles, es decir, progresión desconoce qué estados son realmente relevantes para alcanzar la meta. Luego veremos, que para tratar esta debilidad, se implementa una función heurística que informa a la búsqueda sobre que estados son más relevantes para hallar una camino hacia la meta.

Por el otro lado, una búsqueda regresiva comienza desde la meta de la tarea y regresa sobre acciones para producir submetas, hasta alcanzar una submeta que sea contenida en el estado inicial de la tarea. La intuición detrás de la noción de regresión de una submeta g' sobre una acción a, es que la acción a puede hacer valer la submeta g' si logra al menos una parte de g' y además si no contradice nada de g'.

Definición 15 (Regresión) *Sea* $\Pi = (F, A, c, s_0, G)$ *una tarea de planning STRIPS, el espacio de búsqueda por regresión de* Π *está dado por:*

- $start() = G$
- $is\text{-}target(G') = G' \subseteq s_0$
- $succ(G') = \{(a, G'') \mid si\ G'\ es\ regresable\ sobre\ a\ y\ G'' = regr(G', a)\}$

Diremos que G' *es regresable sobre la acción* a *si* $add(a) \cap G' \neq \emptyset$ *y* $del(a) \cap G' = \emptyset$. *Y definimos* $regr(G', a) = (G' \setminus add(a)) \cup pre(a)$ *siempre que* G' *es regresable por* a.

De esta manera, regresión explora estados que son solubles pero también estados no alcanzables. Es decir, regresión desconoce qué estados son relevantes para alcanzar una submeta que este contenida en el estado inicial de la tarea. Luego, veremos que para tratar esta debilidad, se utiliza una función heurística para informar a la búsqueda sobre que estados son relevantes para alcanzar una submeta contenida en el estado inicial.

La mayoría de los trabajos de investigación se enfocaron en búsqueda progresiva ya que ha demostrado tener mejores resultados empíricos. Esto se puede explicar tal vez, en que en una gran variedad de dominios se observa una importante cantidad de estados espurios que no son estados alcanzables desde el estado inicial. Provocando esto que la regresión explore una importante parte del espacio de estados no relevante para la búsqueda de una solución [Alcázar *et al.*, 2014].

Por su parte, progresión también explora estados no relevantes (aquellos no solubles) pero parece que estos en la práctica son menos frecuentes que los no alcanzables, favoreciendo esto la inclinación de la comunidad hacia la progresión. Además, progresión ha permitido fáciles formulaciones de métodos de búsqueda para formalismos más complejos de planning, por ejemplo, aquellos que introducen la noción de números, duraciones, incertidumbre, etc, mientras que la regresión parece presentar una complejidad intrínseca que dificulta esto último.

Básicamente todos los planificadores basados en búsqueda utilizan progresión, y en particular FD que es el planificador que vamos a utilizar en nuestros experimentos. Por lo tanto, en lo que resta del libro, asumiremos siempre búsqueda progresiva. Aunque para ser justos hay que aclarar que el buen funcionamiento de una u otra puede depender del dominio y del algoritmo de búsqueda particular. Actualmente, no hay un profundo y acabado entendimiento sobre esto dentro de la comunidad de planning. Inclusive existen trabajos recientes que han considerado reutilizar regresión interpolando los avances de la última década producto de la progresión [Alcázar and Torralba, 2015].

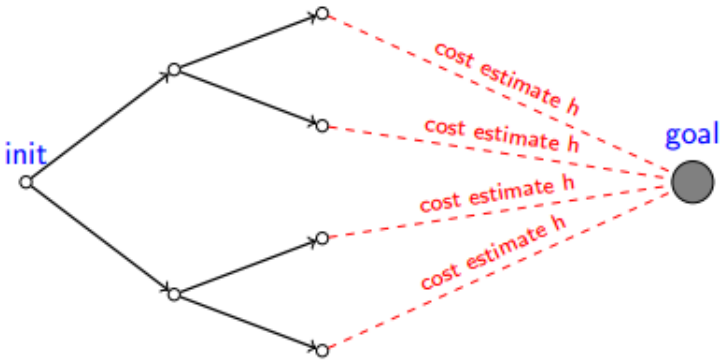


Figura 2.5: Una heurística es simplemente un estimador de la distancia a la meta

Más allá de esto último, como ya mencionamos, nos enfocamos en búsqueda progresiva. Para que la búsqueda por progresión sea más efectiva, necesitamos información respecto de que tan relevantes son los estados alcanzables generados por la búsqueda para llegar a la meta. Para ello, la idea es predecir con la mayor exactitud posible si un estado es soluble y en caso de serlo determinar su nivel de relevancia, o sea, que tan cercano es a la meta. Para ello, se utiliza una función heurística h que estima la distancia de cualquier estado a la meta. Luego la búsqueda prioriza explorar aquellos estados con valor más pequeño de h.

Definición 16 (Función Heurística) *Sea* Π *un problema de planning y sea* $\Theta_\Pi = (S, L, T, I, F)$ *el espacio de estados* Π. *Un función heurística para* Π *es una función* $h : S \to \mathbb{R}_0^+ \cup \{\infty\}$. *El valor* $h(s)$ *es referido*[3] *como el valor heurístico del estado* s.

La figura 2.7 intenta dejar en claro que un heurística no es más que una estimación de la distancia a la meta. Notar que las funciones heurísticas son totalmente arbitrarias, por ejemplo, podemos tomar la heurística que cuenta la cantidad de símbolos proposicionales de la meta que el estado aún no satisface, i. e., $h(s) = |g \setminus s|$. Otro ejemplo, aún más sencillo e inclusive trivial, es tomar la heurística que asigna a cada estado un valor constante i. e., $h_c(s) = c$, para algún $c \in \mathbb{N}$. Esta heurística llamada heurística constante no distingue estados, por lo tanto, no brinda ningún tipo de información a la búsqueda. En la práctica queremos heurísticas que sean lo más exactas posibles en su estimación de la distancia real a la meta. Se dice que una heurística es más informada mientras mayor es su exactitud de estimación.

[3]También referido como el h-valor del estado s.

En particular, existe una heurística llamada *heurística perfecta* que es la más informada de todas.

Definición 17 (Heurística Perfecta) *Sea π un problema de planning y sea Θ_π su espacio de estados. La heurística perfecta de π, denotada por h^*, asigna a cada estado $s \in \Theta_\pi$, el costo de un path óptimo de s a un estado final de Θ_π si s es soluble ó ∞ en caso contrario.*

Además de la precisión en el valor heurístico que se predice, necesitamos que el tiempo de cómputo de la heurística sea el menor posible. Pero estas dos cuestiones entran en conflicto, constituyendo un trade-off entre tiempo de cómputo y calidad de estimación. Por ejemplo, la heurística constante h_c es la menos informada pero trivial de computar, mientras que h^* es la más informada pero con un costo de cómputo elevado, ya que, computar h^* es equivalente a encontrar un plan óptimo del problema y sabemos que esto es PSPACE-completo. Por lo tanto, en la práctica el desafío consiste en hallar heurísticas que aproximen a h^* con el menor costo computacional posible.

También, se buscan heurísticas que cumplan con ciertas propiedades que suelen ser muy interesantes porque permiten garantizar ciertos resultados del algoritmo de búsqueda como optimalidad. Entre estas propiedades se encuentran:

- **Segura**: si $h(s) = \infty$, entonces $h^*(s) = \infty$.
- **Meta-consistente**: $h(s) = 0$, entonces $h^*(s) = 0$.
- **Admisible**: $h(s) \leq h^*(s)$.
- **Consistente**: $h(s) \leq h(s') + c(a)$ siempre que $s \xrightarrow{a} s'$.

La primera propiedad nos asegura que si la heurística indica que un estado es no soluble, entonces realmente lo es. La propiedad de meta consistencia dice que si la heurística nos indica que el estado es final entonces realmente lo es. La propiedad de admisibilidad indica que la heurística siempre subestima la distancia a la meta, i. e., es optimista en su estimación. Por último, la consistencia nos garantiza que una valor heurístico decrece de un estado a su sucesor en a lo sumo el costo de la acción $c(a)$. Se sabe que si h es admisible, entonces h es segura y meta-consistente. Y que si h es consistente y meta-consistente, entonces h es admisible.

En planning, las heurísticas son automáticamente derivadas desde la descripción de la tarea. Actualmente, el estado del arte con respecto a heurísticas para planning se pueden clasificar en cinco diferentes familias [Helmert and Domshlak, 2009]: relajación por deletes, caminos críticos, landmarks y abstracciones.

Relajación por Deletes. Las heurísticas basadas en relajación por deletes (delete-relaxation) (h^+) [Hoffmann and Nebel, 2001a] estiman el costo de resolver una tarea Π como el costo óptimo de un plan de la tarea relajada obtenida a partir de Π eliminado los efectos negativos, i. e., sin considerar la lista *del* de las acciones. Resultados teóricos y experimentales han demostrado que h^+ es una heurística muy informada para un gran número de dominios planning [Hoffmann, 2005; Helmert and Mattmüller, 2008]. Desafortunadamente, computar h^+ en general es NP-completo, lo cual hace inviable el uso directo de h^+ para guiar el proceso de búsqueda. En lugar de ello, se utilizan otras heurísticas polinomiales no admisibles que aproximan h^+ tales como la heurística aditiva (h^{add}) [Bonet and Geffner, 2001], la heurística FF (h^{FF}) [Hoffmann and Nebel, 2001a] y otras admisibles como la heurística máxima ($h^{\max}$) [Bonet and Geffner, 2001] y la heurística LM-cut ($h^{\text{LM-cut}}$) [Helmert and Domshlak, 2009].

Caminos Críticos. Las heurísticas h^m [Haslum and Geffner, 2000] estiman el costo de alcanzar un conjunto de hechos G por medio del más costoso subconjunto de G de tamaño m. Estas heurísticas pueden ser computadas por una análisis de alcanzabilidad considerando tuplas de hechos de cardinalidad a lo sumo m. También es posible computar h^m para una tarea Π a través de la heurística $h^{\max}$ sobre una tarea modificada Π^m, cuyos hechos representan tuplas de hechos de la tarea original Π [Haslum, 2009]. La complejidad de computar h^m es exponencial en m pero polinomial de grado m en el tamaño de la tarea de planning para un m fijo. Se ha demostrado que $h^1 = h^{\max}$ y que $h^m = h^*$ cuando m es igual a la cantidad de variables FDR de la tarea. Más aún, $h^i \leq h^{i+1}$ para todo $i \leq m - 1$. De este modo, el parámetro m establece un trade-off entre la calidad de la estimación de la heurística y su costo computacional. En la práctica h^2 es la más utilizada, ya que, para valores más grandes de m resulta en heurísticas muy costosa de computar.

Landmarks. Un *landmark* es un hecho que debe ser cierto en algún punto de cualquier plan. En satisficing planning han sido utilizados como submetas de la búsqueda [Porteous *et al.*, 2001]. También como heurísticas no admisibles [Richter *et al.*, 2008]. Algunos ejemplos son h^{LA}, una heurística basada en en partición de costos (cost-partitioning) [Karpas and Domshlak, 2009], y LM-CUT ($h^{\text{LM-cut}}$) [Helmert and Domshlak, 2009]. La heurística $h^{\text{LM-cut}}$ está basada en el concepto de *disjuntive action landmark*, un conjunto de acciones tal que en cualquier plan óptimo de la tarea al menos una de las acciones del conjunto debe ser ejecutada. A pesar de que las heurísticas basadas en landmarks fueron inspiradas por ideas diferentes a la detele-relaxation, final-

mente varias de las heurísticas landmarks resultaron ser una estimación de h^+ [Bonet and Helmert, 2010].

Abstracciones. Estas heurísticas transforman una tarea de planning en otra más simple (tratable) y usan una plan de esta última para la estimación de la heurística. Esta transformación consiste en mapear el espacio de estados de la tarea original a un espacio de estados abstracto, donde cada estado es un conjunto de estados de la tarea original. Las transiciones son preservadas por los estados abstractos asegurando la admisibilidad de las heurísticas obtenidas. Existen diferentes métodos para realizar la abstracción del espacio de estados, tales como, Pattern Databases (PDBs) [Culberson and Schaeffer, 1998; Edelkamp, 2001; Haslum *et al.*, 2007], Merge-and-Shrink (M&S) [Helmert *et al.*, 2014] y Cartesian Abstraction [Seipp and Helmert, 2013].

En conclusión, en los últimos años ha habido una gran esfuerzo por parte de la comunidad de planning orientado al desarrollo de heurísticas lo más informadas posibles y computacionalmente tratables. En particular, nuestro trabajo de optimización y principalmente la técnica de subdominización utiliza la relajación por deletes.

2.8. El mundo relajado

Cuando relajamos un problema, descartando o ignorando alguna parte del mismo, obtenemos un nuevo problema a priori más fácil de resolver que nos puede brindar información sobre cómo resolver el problema original (no relajado). En general, la idea de relajación consiste en, dado un problema de planning π cuya heurística perfecta es h^*_π, construir un problema de planning π', vía alguna relajación de π, cuya heurística perfecta $h^*_{\pi'}$ sea útil para estimar h^*_π. En particular, la *relajación por deletes* es un relajación que consiste en eliminar los efectos negativos de las acciones del problema. Esta relajación es ampliamente utilizada en planning por la siguiente propiedad: *si un hecho se vuelve válido, entonces permanecerá válido para siempre.*

Definición 18 (Relajación por Deletes) *(i) Para una acción STRIPS a, con a^+ denotamos la accion relajada por delete, definida por $pre(a^+) = pre(a)$, $add(a^+) = add(a)$ y $del(a^+) := \emptyset$. (ii) Para un conjunto A de acciones STRIPS, con A^+ denotamos el conjunto de acciones relajadas por delete $A^+ = \{a^+ \mid a \in A\}$. Similarmente, para una secuencia de acciones STRIPS $\vec{a} = a_1, \ldots, a_n$, con $\vec{a}^+$ denotamos la secuencia de acciones relajadedas por delete $\vec{a}^+ = a_1^+, ..., a_n^+$. (iii) Para una tarea de planning STRIPS*

$\Pi = (P, A, c, I, G)$, *con* $\Pi^+ = (P, A^+, c, I, G)$ *denotamos la tarea relajada por deletes.* [4]

Se puede observar que la técnica de relajación por deletes es independiente del dominio, i. e., es aplicable a cualquier problema de planning en particular. Por otro lado, como las acciones no tiene la capacidad de eliminar símbolos proposicionales del estado actual, entonces una vez que un símbolo comienza a valer, entonces vale para siempre. Esto implica que en la relajación, por ejemplo, un objeto pueda estar en varias localizaciones al mismo tiempo, ya que, la ubicaión anterior del objeto nunca puede ser borrrada. Obviamentente, que esta característica de la relajación no se ajusta perfectamente al problema real, en el cual, un objeto sólo puede estar en una única ubicación. Pero esta pérdida de información es justamente lo que hace que la relación sea más sencilla de resolver y ha demostrado en la práctica que puede resultar muy útil a la hora de extaer información que luego es utilizada para el resolver el problema no relajado.

Definición 19 (Plan Relajado) *Sea* $\Pi = (P, A, c, I, G)$ *una tarea de planning STRIPS y sea* $s \in$. *Un plan relajado (óptimo) para* s *es un plan (óptimo) para* $\Pi_s^+ = (P, A^+, c, s, G)$, *donde* $\Pi_s = (P, A, s, I, G)$. *Un plan relajado para* I *es un plan relajado para* Π.

A medida que se ejecutan las acciones en el mundo relajado, cada estado contiene al menos los símbolos proposicionales de su predecesor, i. e., los estados del mundo relajado son monótono crecientes.

Definición 20 (Dominancia) *Sea* $\Pi = (P, A, c, I, G)$ *una tarea de planning STRIPS, y sean* s, $s' \in \Theta_\pi$. *Decimos que* s' *domina a* s *si* $s \subseteq s'$.

Proposición 3 *Sea* $\Pi = (P, A, c, I, G)$ *una trea de planning STRIPS, y sean* s, $s' \in \Theta_\pi$, *tal que,* s' *domina a* s. *Entonces:*

1. *Si* s *es una estado meta, entonces* s' *es un estado meta.*
2. *Si* $a \in A$ *es aplicable en* s, *entonces* a *es aplicable en* s', *y además* $R(s', a)$ *domina a* $R(s, a)$.
3. *Si* $\vec{a}$ *es aplicable en* s, *entonces* $\vec{a}$ *es aplicable en* s', *y además* $R(s', \vec{a})$ *domina a* $R(s, \vec{a})$.

[4] Siempre usamos el supra-índice "+" para referirnos a la relajación por deletes de un problema de planning.

Demo 5 *1 Trivial. 2 directo de la definición de R. 3 por inducción en el largo de $\vec{a}$. Sea n el largo de $\vec{a}$. Si $n = 1$ vale por 2. El caso inductivo es directo por hipótesis inductiva y definición de la función R.*

De la proposción anterior podemos concluir que siempre es mejor tener más símbolos proposicionales ciertos.

Proposición 4 *Sea $\Pi = (P, A, c, I, G)$ una trea de planning STRIPS. Sea $s \in \Theta_\pi$ y $a \in A$, entonces $R(s, a^+)$ domina a s y a $R(s, a)$.*

Demo 6 *Directo de la definición de R y a^+.*

Por lo tanto, un plan relajado óptimo estima admisiblemente el costo de un plan óptimo.

Proposición 5 (Admisibilidad de la Relajación por Deletes) *Sea $\Pi = (P, A, c, I, G)$ una tarea de planning STRIPS. Sea $s \in \Theta_\pi$ y $\vec{a}$ un plan de Π_s, entonces $\vec{a}^+$ es un plan relajado de s.*

Demo 7 *Por inducción en el largo de $\vec{a}$ se verifica que $R(s, \vec{a}^+)$ domina $R(s, \vec{a})$. El caso base es trivial, mientras que el caso inductivo es directo de la proposición anterior.*

Por todo lo anterior, tenemos una forma clara de encontrar un plan relajado. Al aplicar acciones relajadas estas sólo pueden hacer más símbolos proposicionales ciertos, entonces esto no puede convertir a un problema de planning soluble en un problema no soluble (por la proposición 3). Por lo tanto, sólo hay que aplicar acciones relajadas hasta que la meta sea alcanzada. Este procedimiento puede ser descripto por el algoritmo 1.

Proposición 6 *El algoritmo 1 es un algoritmo correcto, completo y termina en tiempo polinomial en el tamaño de Π.*

Demo 8 *Veamos correctitud: si $\vec{a}^+$ es devuelto, entonces por construcción $G \subseteq R(s, \vec{a}^+)$. Veamos completitud: Si "Π_s^+ es no soluble" es devuelto por el algoritmo, entonces no existe plan relajado para s^+ y como s^+ domina a s, por proposición 3 esto implica que no pude existir un plan para s. Por último, veamos terminación: cada acción $a^+ \in A^+$ sólo puede ser seleccionada una única vez, ya que, $R(s^+, a^+) = s^+$ siempre que a^+ haya sido previamente aplicada, por lo tanto, el algoritmo termina en a lo sumo $|A^+|$ ciclos.*

Algorithm 1: Algoritmo Voraz de Planning para $\Pi_s^+ = (P, A^+, c, s, G)$.

1: $s^+ := s$
2: $\vec{a}^+ :=<>$
3: **while** $G \not\subseteq s^+$ **do**
4: **if** $\exists a^+ \in A^+$ tal que $pre(a^+) \subseteq s^+ \wedge add(a^+) \not\subseteq s^+$ **then**
5: seleccionar tal a^+
6: $s^+ := R(s^+, a^+)$
7: $\vec{a}^+ := \vec{a}^+ \circ a^+$
8: **else**
9: devolver Π_s^+ es no soluble
10: **end if**
11: **end while**
12: **return** $\vec{a}^+$

En definitiva, es fácil decidr si un plan relajado existe. Es por esto, que el algoritmo 1 resulta útil para generar una función heurística h para guiar la búsqueda en el espacio de estados del problema no relajado: para $s \in \Theta_\pi$ durante la búsqueda progresiva, correr el algoritmo 1 sobre Π_s^+. Luego, asignar a $h(s)$ el costo de $\vec{a}^+$ ó ∞ en caso de que "Π_s^+ es no soluble" fue devuelto por el algoritmo. Por otro lado, esta heurística está lejos de ser exacta i. e., $h \neq h^*$, ya que el algoritmo en cada ciclo selecciona acciones en forma arbitraria, muchas de la cuales pueden ser no relevantes o prescindibles para llegar a la meta, y de esta manera estará sobre-estimando dŕasticamente la verades distancia minima a la meta. En otras pablabras, para ser más exacta, una heurística necesita aproximar la menor distancia a la meta.

Definición 21 (Heurística h^+) *Sea Π una tarea de planning STRIPS y Θ_π su espacio de estados. Entonces la heurística por relajación de deletes óptima h^+ es la función heurística que asigna a cada $s \in \Theta_\pi$ el costo de un plan relajado óptimo de s.*

Proposición 7 *h^+ es admisible, consistente, segura y meta-consistente.*

La heurística h^* satisface las cuatro propiedes deaseables, pero desafortunademnte es intratable.

Definición 22 ($PLANOPT^+$) *Sea $PLANOPT^+$ el problema de decidir si, dado un problema de planning Π y $k \in \mathbb{R}_0^+$, existe un plan relajado de π cuyo costo es a lo sumo k.*

Notar que si tenemos un algoritmo que computa h^*, entonces tenenemos un algoritmo que computa $PLANOPT^+$. Por lo tanto, h^*es al menos tan difícil como $PLANOPT^+$ que ya es intratable.

Teorema 4 *$PLANOPT^+$ es NP-completo.*

Es decir que en la práctica sólo podremos computar heuristicas que aproximen h^+. En planning, heurísticas *no admisibles* surgen como aproximaciones a heurísticas admisibles que son muy costosas de computar.

Capítulo 3

El Proceso de Grounding

3.1. Predicados y Esquemas de Acción

Todos los conceptos básicos de planning desarrollados en el capítulo 2 fueron realizados en términos proposicionales. Sin embargo, a la hora de describir una tarea de planning, el paradigma proposicional presenta dificultades inherentes. Por ejemplo, supongamos que queremos modelar la ubicación de un agente A_1 y nuestro mundo tiene n ubicaciones. Entonces vamos a tener que especificar n símbolos proposicionales p_i, tal que, p_i denota "el agente A_1 esta en la ubicación i". ¿Qué sucede si ahora además tenemos un agente A_2?, debemos especificar n símbolos proposicionales nuevos para modelar la ubicación del agente A_2. ¿O que sucede si ahora tenemos nuevos ubicaciones?, también tenemos que especificar nuevos símbolos proposicionales para modelar estas nuevas ubicaciones.

La ubicación de un agente es independiente del agente. Por lo tanto, podemos modelar la ubicación de un agente en forma más general y compacta, a través de un predicado de la forma $at(x, y)$ que denota que el agente x se encuentra en la ubicación y. Al ser x e y variables, el predicado $at(x, y)$ codifica un conjunto de símbolos proposicionales, en particular, uno por cada agente y ubicación posible. El predicado $at(x, y)$ es más flexible a las cantidad de agentes y ubicaciones disponibles, debido a su naturaleza esquemática, en comparación con la versión puramente proposicional. Desde el punto de vista del usuario, los predicados con parámetros nos permiten tener una especificación más compacta y versátil para una tarea de planning.

Análogamente, para el caso de las acciones, sucede una situación idéntica. Supongamos que queremos modelar la acción que modifica la ubicación actual del agente. En el paradigma proposicional tenemos que describir una acción por cada agente y par de ubicaciones posibles, presentando los mismos

problemas ya mencionados. Entonces, en lugar de describir acciones proposicionales, resulta más conveniente describir acciones en forma esquemática utilizando variables. Por ejemplo, podemos describir un esquema de acción de la forma $Move(x, y_1, y_2)$ que modela el cambio de ubicación del agente x, desde la ubicación y_1 a la ubicación y_2. Nuevamente, el esquema de acción denota un conjunto de acciones, una por cada agente y par de ubicaciones posible. De esta manera, obtenemos una representación más compacta y flexible de las acciones de nuestra tarea de planning. En definitiva, resulta conveniente (desde el punto de vista del usuario) poder especificar una tarea de planning en términos de predicados y esquemas de acción, y no en términos de símbolos proposicionales (facts) y acciones.

Definición 23 *Un predicado es un símbolo atómico de la forma $p[X]$, con X un conjunto de variables, llamada interfaz. Definimos la aridad de $p[X]$ como la cantidad de variables en su interfaz ($|X|$).* [1]

Denotaremos con $p(x_1 := o_1, \ldots, x_n := o_n)$ al símbolo proposicional que se obtiene del predicado $p[x_1, \ldots, x_n]$ al reemplazar toda ocurrencia de x_i por el objeto o_i. En este caso, decimos que el símbolo proposicional $p(x_1 := o_1, \ldots, x_n := o_n)$ deriva del predicado $p[X]$.

Por otra parte, si la fórmula que ocurre en la precondición y en el efecto de un esquema de acción es una conjunción de literales, pero donde los símbolos atómicos son predicados (en lugar de símbolos proposicionales), decimos que se trata de un esquema de acción STRIPS.

Definición 24 *Formalmente, un esquema de acción STRIPS es una 3-upla $a[X] = (pre, del, add)$ donde pre, del y add son conjuntos de predicados, llamado precondición, "lista del" y "lista add", respectivamente. X es el conjunto de variables que ocurren en $pre \cup del \cup add$, llamada interfaz de $a[X]$. Definimos la aridad de $a[X]$ como la cantidad de variables en su interfaz ($|X|$).*

Denotaremos con $a(x_1 := o_1, \ldots, x_n := o_n)$ a la acción proposicional que se obtiene del esquema de acción $a[x_1, \ldots, x_n]$ al reemplazar toda ocurrencia de x_i por el objeto o_i. En este caso, decimos que la acción proposicional $a(x_1 := o_1, \ldots, x_n := o_n)$ deriva del esquema $a[X]$.

Si bien describir la tarea de planning en forma esquemática resulta muy conveniente desde el punto de vista del usuario, los planificadores realizan la

[1] Las variables de los predicados se listan entre corchetes, es decir, $p[x_1, \ldots, x_n]$ con x_i variables ó $p[X]$ con X (mayúscula) cuando el nombre de las variables individuales no es relevante. Esta misma notación vale para esquemas de acción.

búsqueda de un plan sobre una representación proposicional de la tarea. Esto se debe a que las técnicas de búsqueda se ejecutan más naturalmente bajo esa representación. Es por esto, que surge la necesidad de un proceso de conversión de la tarea desde su versión esquemática a su versión proposicional. Este proceso recibe el nombre de grounding y tiene lugar en el planificador como una etapa de pre-procesamiento a la etapa de búsqueda. Este proceso es lo que vamos a estudiar en detalle en este capítulo, ya que, resulta ser el blanco de nuestras dos técnicas de optimización.

El proceso de grounding consiste en obtener la representación proposicional de una tarea de planificación, para conjuntos de predicados y esquemas de acción. Consideremos nuevamente el predicado $at(x, y)$, que modela la ubicación actual (y) del agente x. Si en nuestro mundo, tenemos agentes A_1, $A2$ y ubicaciones u_1, u_2 el grounding del predicado $at(x, y)$ resulta ser el conjunto de símbolos proposicionales $\{at(A_1, u_1), at(A_1, u_2), at(A_2, u_1), at(A_2, u_2)\}$. A su vez, el grounding del esquema de acción $Move(x, y_1, y_2)$ resulta en el conjunto de acciones proposicionales:

$$\begin{array}{ll} \{ & Move(A_1, u_1, u_1),\ Move(A_1, u_1, u_2), \\ & Move(A_1, u_2, u_1),\ Move(A_1, u_2, u_2), \\ & Move(A_2, u_1, u_1),\ Move(A_2, u_1, u_2), \\ & Move(A_2, u_2, u_1),\ Move(A_2, u_2, u_2) \ \ \}. \end{array}$$

Además, si el esquema $Move(x, y_1, y_2)$ esta definido de la siguiente manera:

$$\begin{array}{l} Action : Move(x, y_1, y_2) \\ pre : at(x, y_1) \\ del : at(x, y_1) \\ add : at(x, y_2) \end{array}$$

Entonces la acción proposicional $Move(A_1, u_1, u_2)$ que se obtiene mediante el reemplazo de variables $x := A_1, y_1 := u_1, y_2 := u_2$, queda definida de la siguiente manera:

$$\begin{array}{l} Action : Move(A_1, u_1, u_2) \\ pre : at(A_1, u_1) \\ del : at(A_1, u_1) \\ add : at(A_1, u_2) \end{array}$$

Notar, que luego del reemplazo sintáctico, la ocurrencia de $at(A_1, u_1)$ en la precondición de la acción $Move(A_1, u_1, u_2)$ es un símbolo proposicional que denota "el agente A_1 se encuentra en la ubicación u_1". Es decir, el reemplazo sintáctico convierte predicados y esquemas de acción (con variables) en

símbolos y acciones proposicionales (sin variables). En definitiva, en una primera fase, el proceso de grounding de un planificador consiste simplemente en realizar reemplazos sintácticos de variables por objetos del mundo para todos los predicados y esquemas de acción definidos. Un símbolo proposicional que resulta de tal reemplazo sintáctico a partir de un predicado decimos que es un predicado instanciado, grounded o simplemente un "hecho" (o fact, en inglés). Al mismo tiempo, una acción que resulte de tal reemplazo sintáctico a partir de esquema de acción decimos que es una acción instanciada, grounded, o simplemente una acción. El mapeo entre variables y objetos del mundo que se usaron para obtener una acción instanciada decimos que es una instanciación. Llamaremos dominio, a un conjunto de predicados y esquemas de acción. Y universo al conjunto de objetos del mundo. Entonces, un primer algoritmo de grounding consiste en tomar cada predicado y esquema de acción del dominio e instanciar sus interfaces en todas las formas posibles con los objetos del universo. Este proceso lo llamaremos *Grounding Cartesiano* y lo describimos con mayor precisión en la siguiente sección.

3.2. Grounding Cartesiano

El concepto de grounding cartesiano consiste en recolectar todas las instanciaciones posibles de predicados y esquemas de acción de un dominio, obtenidas mediante reemplazo sintáctico. Formalmente, lo podemos definir de la siguiente manera:

Definición 25 (Grounding Cartesiano) *Sea D un dominio y O un conjunto de objetos. Entonces, el Grounding Cartesiano (CG) es el conjunto de predicados y acciones instanciadas* $CG = \{e(x_1 := o_1, \dots x_n := o_n) \mid e[x_1, \dots x_n] \in D \text{ y } o_i \in O\}$.

Observar que si un esquema de acción tiene k variables en su interfaz y existen n objetos en el universo, entonces el esquema genera n^k acciones instanciadas (lo mismo ocurre para el caso de predicados). Esto quiere decir que el grounding es exponencial en la aridad de los esquemas de acción y predicados del dominio. En la práctica, esto no representa un gran problema, ya que, en la mayoría de los dominios que se estudian tienen una aridad máxima de cuatro o cinco variables. Sin embargo, se pueden encontrar dominios puntuales, cuya aridad y cantidad de objetos del universo escala en forma tal que convierten el proceso de grounding en una tarea muy difícil de completar por el planificador [Matloob and Soutchanski, 2016]. En particular, nuestra técnica *Action Schema Splitting* logra reducir considerablemente la aridad de los esquemas de acción de un domino, permitiendo completar el proceso

de grounding llevada adelante por el planificador. Es importante notar que una vez fijado el dominio, el proceso de grounding CG es polinomial con respecto a la cantidad de objetos del universo. Este proceso es implementado mediante el algoritmo 2.

Algorithm 2: Proceso de Grounding Cartesiano.

Entrada: D un dominio y O un universo

1: **for each** $p[x_1, \ldots x_n] \in D$ **do**
2: **for each** $(o_1, \ldots o_n) \in O$ **do**
3: $F := F \cup p(x_1 := o_1, \ldots x_n := o_n)$
4: **end for**
5: **end for**
6: **for each** $a[x_1, \ldots x_n] \in D$ **do**
7: **for each** $(o_1, \ldots o_n) \in O$ **do**
8: $A := A \cup a(x_1 := o_1, \ldots x_n := o_n)$
9: **end for**
10: **end for**
11: **return** F y A

En las próximas secciones estudiamos cómo optimizar el resultado del grounding cartesiano, en el sentido de tratar de obtener un subconjunto de sus predicados y acciones instanciadas, tal que, los planes se preserven en una relación uno a uno. Es decir, que la tarea proposicional definida por este subconjunto sea equivalente a la tarea definida por el grounding cartesiano.

3.3. Predicados Estáticos

Durante el proceso de grounding, podemos utilizar cierto conocimiento implícito del dominio para obtener una representación proposicional más pequeña de la tarea, eliminando acciones y símbolos proposicionales sin afectar los planes de la tarea original. En particular, podemos detectar aquellos predicados cuya validez no se modifica a lo largo de toda la tarea de planning. Entre estos predicados, se encuentran aquellos que no ocurren en los efectos de ningún esquema de acción, lo cuales, reciben el nombre de predicados estáticos.

Definición 26 *Diremos que $p[X]$ es un predicado estático si no ocurre en los efectos de ningún esquema de acción del dominio.*

Los predicados estáticos pueden ser utilizados para codificar propiedades que no varían en ningún momento de la tarea, por ejemplo, el tipo de un

objeto o la existencia de una ruta entre dos ciudades. Otro ejemplo, puede ser cuando modelamos el tipo "agente" mediante un predicado de la forma "$is\text{-}agent(x)$" que vale siempre que x es un agente. También modelamos la existencia de una ruta entre la ciudad x e y mediante el predicado "$road(x, y)$".

Un símbolo proposicional puede ser removido de la tarea proposicional en base a la siguiente observación. Sea p un símbolo proposicional definido por un predicado estático $p[X]$. Entonces si p ocurre en el estado inicial, entonces p vale siempre. Por lo tanto, podemos eliminar p del estado inicial, de la meta (si ocurre) y de todas las precondiciones de las acciones. Mientras que si p no ocurre en el estado inicial, entonces p no vale en todo momento. Por lo tanto, podemos eliminar las acciones que tienen a p en su precondición, ya que, estas acciones no son aplicables. Notar que remover de esta forma los símbolos proposicionales definidos por predicados estáticos no altera los planes de la tarea. Para más detalle sobre el tratamiento de predicados estáticos se puede consultar a [Koehler and Hoffmann, 2000].

3.4. Grounding por Alcanzabilidad Relajada

Este proceso de grounding consiste en utilizar la relajación por deletes de una tarea de planning (ver sección 2.8). Esta relajación permite generar una menor cantidad de hechos y acciones proposicionales respecto a las que se generan mediante grounding cartesiano. La idea consiste en solamente generar aquellos hechos y acciones que "ocurren en algún momento" en la tarea relajada, descartando el resto. Este proceso de grounding es el implementado por el planificador Fast-Downward (FD) [Helmert, 2006].

Definición 27 (Alcanzabilidad de hechos y acciones) *Sea* $\Pi = (F, A, s_0, g)$ *una tarea de planning STRIPS,* $p \in F$ *y* $a \in A$*. Entonces, diremos que* p *es alcanzable en* Π *sii existe un estado alcanzable* $s \in \Theta_\Pi$ *tal que* $p \in s$*. Y diremos que* a *es alcanzable en* Π *sii existe un estado alcanzable* $s \in \Theta_\Pi$ *tal que* a *es aplicable en* s*. Además, diremos que* p *es alcanzable relajadamente en* Π *sii* p *es alcanzable en* Π^+ *y que* a *es alcanzable relajadamente en* Π *sii* a^+ *es alcanzable en* Π^+*.*

Sabemos que toda acción y hecho que no es alcanzable relajadamente, entonces no puede ser alcanzable en la tarea original, por lo tanto, esas acciones y hechos se pueden descartar de la representación proposicional de la tarea, sin alterar los planes de la misma. En otras palabras, en este proceso de grounding, las acciones y hechos de la representación proposicional son exactamente aquellas que son alcanzables relajadamente. Afortunadamente,

decidir si un hecho o una acción es relajadamente alcanzable es un problema sencillo desde el punto de vista computacional.

Proposición 8 *Sea $REACHABILITY_+^+$ el problema de decidir si un hecho o una acción es relajadamente alcanzable, entonces $REACHABILITY_+^+ \in P$. Más aún, decidir alcanzabilidad relajada es lineal en la cantidad de acciones.*

Demo 9 *Este resultado es una consecuencia directa de la demostración de que $PLANSAT_+^+ \in P$ (ver sección 2.4). Simplemente, en cada iteración del algoritmo chequeamos lo siguiente: en el caso de un hecho si este ocurre en el estado correspondiente a la última iteración y en el caso de una acción si los hechos de su precondición ocurren en este último estado alcanzado. En el peor caso, hacemos tantas iteraciones como acciones posibles. Por lo tanto, decidir alcanzabilidad relajada es lineal en la cantidad de acciones.*

Este proceso de grounding que genera solamente aquellos hechos y acciones proposicionales que son alcanzables relajadamente, se puede implementar fácilmente, utilizando una cola inicializada con los hechos del estado inicial de la tarea. En cada iteración, se toma de la cola un elemento que puede ser un hecho o una acción:

- Si el elemento es un hecho, se lo marca como procesado y se agregan a la cola todas las acciones (aún no procesadas) tal que son aplicables bajo los hechos procesados hasta el momento.
- Si el elemento es una acción, se marca como procesada y agregamos a la cola todos sus efectos (hechos) positivos. Este procedimiento se repite hasta que la cola se encuentre vacía, lo cual indica que todos los hechos y acciones alcanzables relajadamente fueron procesados [Helmert, 2009].

Sin embargo, un hecho o una acción que es relajadamente alcanzable puede no ser alcanzable en la tarea original. Es justamente esta pérdida de información la que origina que computar en el mundo relajado sea polinomial. Más aún, puede ocurrir que muchos de estos hechos y acciones relajadamente alcanzables sean efectivamente alcanzables en la tarea original pero al mismo tiempo pueden ser irrelevantes para alcanzar la meta. Es decir, pueden ser acciones que no forman parte de ningún plan de la tarea. Es aquí, donde nuestra segunda técnica de optimización llamada Subdominización [Gnad *et al.*, 2019] intenta predecir cuál de estos hechos y acciones son verdaderamente relevantes para alcanzar la meta, priorizando el procesamiento de las acciones más relevantes por sobre el resto. En definitiva, la técnica de subdominización es una optimización del proceso de grounding por alcanzabilidad relajada. Los detalles de esta técnica se estudian en detalle en la parte III de este libro.

3.5. Esquemas de Acción ADL

Como consecuencia del uso predicados y esquemas de acción, tenemos la posibilidad de utilizar fórmulas de la lógica de predicados para describir fórmulas y efectos ADL de una forma compacta. Las fórmulas y los efectos en esquemas ADL tienen los operadores proposicionales habituales ($\wedge$, $\vee$, $\neg$, $\Rightarrow$), pero los símbolos atómicos son predicados de la forma $p[X]$, en lugar de simples símbolos proposicionales (sin variables). Además, lo más interesante de la lógica de predicados es que permite la noción de cuantificación de variables.

Supongamos que queremos expresar que algún agente del mundo A_i se encuentra en la ubicación u_1. Esto se puede expresar en lógica de predicados mediante la cuantificación $\exists x : at(x, u_1)$, lo cual se corresponde, en un mundo con n agentes, a la fórmula proposicional ADL $at(A_1, u_1) \vee \cdots \vee at(A_n, u_1)$. O supongamos, queremos expresar que todos los agentes del mundo se encuentran en la misma ubicación u_1. Esto se puede expresar mediante la cuantificación $\forall x : at(x, u_1)$ que se corresponde con la fórmula proposicional ADL $at(A_1, u_1) \wedge \cdots \wedge at(A_n, u_1)$. De esta forma, las cuantificaciones permiten expresar disyunciones y conjunciones proposicionales ADL en forma compacta.

El concepto de cuantificación, trae aparejado los conceptos de variable libre y ligada. Una variable se dice libre si no está alcanzada por un cuantificador, caso contrario, se dice que está ligada o cuantificada. Con $\varphi(x)$ denotamos que x es una variable libre de la fórmula φ. En general, la cuantificación existencial $\exists x : \varphi(x)$ se corresponde con la disyunción proposicional finitaria $\varphi(o_1) \vee \cdots \vee \varphi(o_n)$. Análogamente, la cuantificación universal $\forall x : \varphi(x)$ se corresponde con la conjunción proposicional finitaria $\varphi(o_1) \wedge \cdots \wedge \varphi(o_n)$, con o_i objetos del mundo. Además, podemos cuantificar universalmente sobre los efectos condicionales de un esquema de acción ADL. Sea $\varphi(x) \Rightarrow l(x)$ un efecto condicional con φ una fórmula y l un literal de la lógica de predicados. Entonces la cuantificación de un efecto condicional $\forall x : \varphi(x) \Rightarrow l(x)$ representa el efecto condicional proposicional ADL $\varphi(o_1) \Rightarrow l(o_1) \wedge \cdots \wedge \varphi(o_n) \Rightarrow l(o_n)$.

3.6. El lenguaje PDDL

Para especificar un tarea de planning en una computadora, es necesario tener un lenguaje de especificación formal amigable para el usuario, donde se describan todos los detalles de la tarea. Para dar respuesta a esta necesidad, se definió PDDL (Planning Domain Description Language) que se ha transformado en el lenguaje de especificación estándar dentro de la comunidad de planning. Este lenguaje fue inicialmente propuesto por Drew Mc

Dermott para la Competencia Internacional de Planning (IPC) de los años 1998 y 2000 [McDermott *et al.*, 1998]. En este evento, se comparan las performances de diferentes planificadores, por lo cual, resultaba muy necesario un lenguaje de especificación común para todos los participantes del evento. El lenguaje PDDL se construye a partir de la lógica de predicados, lo cual permite representar en forma esquemática tareas proposicionales STRIPS y ADL. PDDL1.2 fue el lenguaje oficial de la primera edición de la competencia IPC y separaba la definición esquemática de una tarea de planning en dos archivos: el dominio y la tarea.

En el archivo dominio se definen los predicados y los esquemas de acción disponibles. Mientras que en el archivo de tarea se definen los objetos concretos del mundo a modelar, el estado inicial y la meta. La separación entre dominio y tarea resulta importante, ya que, diferentes tareas de un mismo dominio comparten una estructura común que es capturada por el dominio.

La mayoría de los benchmarks de planning se encuentran estructurados de esta manera, un dominio y múltiples tareas distintas para ese mismo dominio. En particular, el benchmark utilizado en todos nuestros experimentos son estructurados en términos de un archivo dominio y su conjunto de archivos de tarea en lenguaje PDDL. Como ya mencionamos, este lenguaje describe una tarea de planning en forma esquemática mediante predicados y esquemas de acción. Sin embargo, los planificadores realizan la búsqueda de un plan sobre una representación proposicional de la tarea, de más bajo nivel de abstracción que PDDL y que resulta más conveniente para la aplicación directa de técnicas de búsqueda. Es por esto, que la mayoría de los planificadores transforman la especificación PDDL (de más alto nivel) de una tarea de planning a una representación proposicional STRIPS o funcional (FDR) (de más bajo nivel). Esta transformación se conoce como *Proceso de Grounding* y tiene lugar dentro del planificador como una etapa de pre-procesamiento previa a la etapa de búsqueda, como ya habíamos dicho en la sección 3.1.

La sintaxis de PDDL está inspirada en el lenguaje de programación Lisp, lo cual se refleja, por ejemplo, en la ocurrencia de operadores en forma prefija:

```
(and (or (at A1 u1) (on A2 u1)) (or (on A1 u2) (on A2 u2)))
```

Por su parte, el archivo domino donde se definen los predicados, los esquemas de acción y los tipos se estructura de la siguiente manera:

- definición del nombre del dominio:

  ```
  (define (domain AGENT-DOMAIN)
  ```

- definición de requerimientos:

```
(:requirements :strips :typing)
```

- definición de tipos (las variables pueden ser tipadas):

```
(:types AGENT LOCATION)
```

- definición de los predicados:

```
(:predicates (at ?x - AGENT ?y - LOCATION>))
```

- definición de los esquemas de acción.

```
(:action Move
   :parameters (?x - AGENT ?y1 - LOCATION ?Y2 - LOCATION)
   :precondition (and (at ?x ?y1))
   :effect (and (not (at ?x ?y1)) (at ?x ?y2))))
```

La precondición de un esquema de acción es una fórmula que respeta la siguiente sintaxis:

```
<formula> :=
            |<predicate>
            |(and <formula> ... <formula>)
            |(or <formula> ... <formula>)
            |(not <formula>)
            |(forall (?x1 - type_1 ... ?xn - type_n) <formula>)
            |(exists (?x1 - type_1 ... ?xn - type_n) <formula>)
```

Mientras que el efecto de un esquema de acción debe seguir la siguiente sintaxis:

```
<effect> :=
           |<predicate>
           |(not <predicate>)
           |(and <effect> ... <effect>)
           |(when <formula> <effect>)
           |(forall (?x1 - type_1 ... ?xn - type_n) <effect>)
```

El archivo de la tarea, donde se definen los objetos del mundo, el estado inicial y la meta se estructura de la siguiente forma:

- definición del nombre de la tarea:

```
(define (problem AGENT_TASK)
```

- definición del dominio asociado de la tarea:

```
(:domain AGENT-DOMAIN)
```

- definición de los objetos del mundo:

```
(:objects A1 A2 A3 - AGENT U1 U2 U3 - LOCATION)
```

- definición del estado inicial:

```
(:init at(A1,U1) at(A2,U1) at(A3,U1))
```

- definición de la meta:

```
(:goal (and at(A1,U1) at(A2,U2) at(A3,U3))))
```

Cabe aclarar que el estado inicial es una lista de predicados instanciados (facts) y la meta es una fórmula que sigue la misma sintaxis que las precondiciones de los esquemas de acción.

Nuestra técnica de optimización *Action Schema Splitting* opera sobre la descripción PDDL de una tarea de planning. La técnica consiste en particionar cada esquema de acción en sub-esquemas, las cuales ejecutan una parte del esquema original preservando su semántica. De esta manera logramos reducir considerablemente el tamaño de las interfaces de los esquemas de acción del dominio. Los detalles de esta técnica se estudian a continuación, en la Parte II de este libro.

Parte II

Action Schema Splitting

Capítulo 4

Fundamentos de Splitting

En este capítulo presentamos nuestra primer técnica de optimización del proceso de grounding, la cual intenta reducir la cantidad de acciones proposicionales de una tarea de planning. La técnica llamada *Action Schema Splitting* actúa sobre la representación PDDL de la tarea. El método consiste básicamente en transformar un esquema de acción en un conjunto de sub-esquemas con una menor interfaz con respecto al esquema original. Logrando de esta manera una reducción del número de acciones resultantes durante la instanciación. Algunas de las ideas subyacentes de la técnica ya eran aplicadas manualmente. Sin embargo, nunca antes fueron acabadamente descriptas y estudiadas formalmente por la comunidad de planning y menos aún automatizadas algorítmicamente.

Las transformaciones sobre dominios de planning han sido una temática tradicional de interés dentro de la comunidad [Gazen and Knoblock, 1997; Nebel, 2000; Palacios and Geffner, 2009], debido al impacto en el rendimiento de los planificadores que ha demostrado tener la forma en que se especifica una tarea de planning. Por ejemplo, se ha considerado remover acciones redundantes con el objetivo de reducir el factor de branching de la búsquda [Haslum and Jonsson, 2000]. También se introdujo la idea de macro-acciones con el fin de reducir la distancia a la meta [Botea *et al.*, 2005; Newton *et al.*, 2007], o inclusive simplificar la tarea preservando ciertas propiedades utilizando el concepto de grafo causal y otros tipos de análisis [Knoblock, 1994; Haslum, 2007]. En definitiva, las transformaciones más interesantes son aquellas que permiten inferir propiedades del dominio original a partir del dominio transformado. Es decir, son transformaciones que no alteran la tarea de planning original respecto de cierta característica determinada, y de esta manera, respecto a dicha característica el dominio y su transformación son equivalentes.

Por nuestra parte vamos a describir una transformación que actúa sobre

los esquemas de acción de un dominio logrando una reducción en el tamaño de las interfaces de los esquemas con el fin de que esto origine una disminución de las acciones proposicionales que se generan en el proceso de grounding. Esta transformación implica un tade-off entre la reducción del tamaño de la interfaz de los esquemas del dominio (y como consecuencia el número de acciones proposicionales que se generan) y la longitud de los planes, ya que, la introducción de sub-esquemas en lugar de los esquemas originales tiene el efecto no deseado de aumentar la distancia a la meta. Nuestros experimentos muestran que esta técnica puede resultar exitosa principalmente en aquellos dominios donde los esquemas de acción se caracterizan por tener interfaces de gran tamaño. Introduciremos la técnica de splitting sobre esquemas de acción STRIPS y luego la extenderemos a esquemas de acción ADL, donde reutilizaremos para ello la mayoría de las ideas y conceptos utilizados en el caso de esquemas STRIPS.

4.1. Ejemplo, motivación y dificultades de un Splitting

Dado un esquema de acción STRIPS $a[X]$, la operación de splitting crea sub-esquemas $a_1[X_1], \ldots, a_n[X_n]$ donde cada sub-esquema ejecuta una parte del esquema original y cada uno ellos posee una interfaz más pequeña con respecto a la interfaz del esquema original. Los sub-esquemas son tales que la ejecución de los mismos tiene el mismo efecto sobre un estado arbitrario que la ejecución del esquema original. De esta manera, la ventaja principal de un splitting es la reducción del número de acciones proposicionales que se generan a partir de los esquemas de acción de un dominio. Por ejemplo, si cada variable $x \in X$ puede ser instanciada con 100 objetos, $|X| = 3$ y $|X_i| = 1$, entonces el esquema original $a[X]$ generara 1 millón de acciones proposicionales contra las 300 que genera el splitting del esquema.

En el caso de un dominio STRIPS, las precondiciones y los efectos son conjunciones de literales, donde los símbolos atómicos son predicados de la forma $p[X]$. Esto se puede representar mediante tres conjuntos de símbolos atómicos.[1] Por ejemplo, consideremos el siguiente esquema de acción STRIPS que modela una acción que mueve un bloque x, que se encuentra arriba de un bloque y, sobre un bloque z:

[1] A los predicados también los llamaremos *"átomos"*.

$$
\begin{array}{l}
\boldsymbol{Move}(x, y, z) \\
pre : \{on(x, y), clear(x), clear(z)\} \\
del : \{on(x, y), clear(z)\} \\
add : \{on(x, z), clear(y)\}
\end{array}
$$

El predicado $on(x, y)$ modela la situación donde el bloque x se encuentra colocado arriba del bloque y, mientras que el predicado $clear(x)$ modela la situación en la que ningún bloque se encuentra encima del bloque x. La operación de splitting sobre un esquema de acción consiste en particionar los predicados que ocurren en su precondición y efecto. En el ejemplo 1, se muestra un splitting tentativo para el esquema $Move(x, y, z)$, el cual, nos resultará muy útil para introducir las dificultades que presenta la técnica.

Ejemplo 1 *Splitting tentativo del esquema* $Move(x, y, z)$.

$$
\begin{array}{ll}
\boldsymbol{Move}_1(x, y) & \boldsymbol{Move}_2(x, z) \\
\quad pre : \{on(x, y), clear(x)\} & \quad pre : \{clear(z)\} \\
\quad del : \{on(x, y)\} & \quad del : \{clear(z)\} \\
\quad add : \{clear(y)\} & \quad add : \{on(x, z)\}
\end{array}
$$

En el ejemplo, se observa que cada sub-esquema ejecuta una parte del esquema original y en forma única. Además, los sub-esquemas se pueden ejecutar en cualquier orden sin afectar el estado resultante. Es decir, si aplicamos $Move_1(x, y)$ y después $Move_2(x, z)$, tenemos el mismo resultado final que si aplicamos primero $Move_2(x, z)$ y luego $Move_1(x, y)$. Esto significa que el simple particionamiento de un esquema no impone un orden de ejecución entre sus partes. A su vez, cada sub-esquema presenta una interfaz más pequeña en comparación al esquema original, lo cual, implica que el número de acciones proposicionales que genera el splitting es menor al que genera el esquema original. Si tenemos n bloques en el mundo, entonces el esquema original genera n^3 acciones proposicionales con las cuales finalmente se realiza el proceso de búsqueda. Mientras que el splitting genera $2n^2$ acciones instanciadas. Para valores grandes de n, esto representa una importante reducción de las acciones instanciadas. Por otra lado, a primera vista, la correspondencia del splitting con el esquema original parece obvia. Sin embargo, el splitting mostrado no es correcto debido a que no asegura las siguientes tres propiedades:

1. **Ejecución en bloque:** nada asegura que los sub-acciones pertenecientes a un mismo splitting son aplicadas todas y consecutivamente. Puede ocurrir que alguna otra acción es aplicada entre medio de dos sub-acciones de un mismo splitting. En el ejemplo, luego de aplicar la sub-acción $Move_1(x, y)$ se puede ejecutar una acción distinta a

$Move_2(x, z)$, provocando de esta manera un interleaving entre las sub-acciones del splitting y otras acciones del dominio. También, puede ocurrir una ejecución parcial del splitting, es decir, que sólo se ejecuten sub-acciones correspondientes a un tramo inicial del splitting, faltando aplicar el tramo final que completaría la ejecución total del mismo. En el ejemplo, luego de aplicar la sub-acción $Move_1(x, y)$ nada garantiza que se ejecute tarde o temprano la sub-acción $Move_2(x, y)$. En conclusión, una vez que comienza la ejecución de un splitting debemos evitar interleaving con acciones que no pertenecen al mismo. Y además, evitar ejecuciones incompletas, es decir, el spltting debe ejecutarse siempre en forma atómica.

2. **Instanciación consistente:** nada asegura que las variables comunes de los sub-esquemas de un splitting son instanciadas al mismo objeto. En el ejemplo, la variable compartida x debe ser instanciada al mismo objeto en ambos sub-esquemas. En caso contrario, se puede ejecutar la sub-acción $Move_1(x, y)$ y luego $Move_2(x, z)$ pero donde la variable x se instanció de manera diferente en cada sub-acción. Esta dificultad se origina como consecuencia de particionar la interfaz del esquema original, ya que, ahora las variables compartidas pasan a ser variables distintas para cada sub-esquema en particular, a pesar de tener el mismo nombre. Por lo tanto, debemos forzar que esas variables distintas pero con el mismo nombre, sean instanciadas al mismo objeto.

3. **Atómicamente bien ordenado:** nada asegura que un predicado que ocurre tanto en la precondición como en el efecto, ocurra en el splitting primero como efecto y luego como precondición. Esto puede ocasionar una inconsistencia con respecto a la ejecución de la acción original. En el ejemplo, si las variables y y z son instanciados al mismo bloque, entonces en cualquier estado alcanzable s el esquema original no es aplicable debido a que no puede valer $on(x, y)$ and $clear(y)$ al mismo tiempo. Sin embargo, en el splitting el efecto positivo $clear(y)$ de $Move_1(x, y)$ hace valer la precondición $clear(z)$ de $\text{Move}_2(x, z)$, provocando que el splitting sea aplicable en s. Esta dificultad tiene su origen en que la operación de splitting rompe la ejecución atómica de los facts de la acción original. Es decir, en el splitting los facts no se ejecutan todos en un mismo instante y esto puede provocar que ciertos efectos eliminen o agreguen precondiciones de la acción misma, ocasionando que el splitting sea aplicable o no, independiente de lo que ocurre con la acción original. En definitva, debemos asegurar que la acción original es aplicable en un estado s si y sólo si su splitting correspondiente es

aplicable en s.

La propiedad (3) es la más sutil y difícil de comprender. Es por eso que, tal vez, requiere de una explicación más detallada. Supongamos que tenemos una acción instanciada a, tal que, $pre(a) = del(a) = \{p\}$ y consideremos el splitting que consiste en dos sub-acciones a_1 y a_2, tales que, $del(a_1) = \{p\}$ y $pre(a_2) = \{p\}$. Si s es un estado en el que a es aplicable, entonces al aplicar a_1 en el estado s provoca que a_2 sea no aplicable en el estado resultante, pues a_1 eliminó el fact p que era necesario para la ejecución de a_2. Por lo tanto, el splitting no es aplicable a partir del estado s mientras que la acción original a si lo era. Como ya mencionamos, la operación de splitting rompe la ejecución atómica de los facts de la acción original y esto puede provocar que el splitting se comporte de forma diferente a como lo haría la acción original. A nivel de los esquemas de acción, esto se presenta cuando un predicado $p[X]$ ocurre en la precondición y en la lista *del* o *add*, ya que, cualquier instanciación de X provoca que un mismo fact ocurra en la precondición y en el efecto de la acción instanciada. Esta dificultad también se presenta más sutilmente cuando un predicado aparece en la precondición y en el efecto del esquema pero con una interfaz distinta. Esas interfaces distintas pueden aún ser instanciadas a los mismos objetos, provocando de esta manera la aparición de un mismo fact en la precondición y en el efecto de una acción instanciada. Por ejemplo, supongamos que un predicado $p[x]$ ocurre en la precondición del esquema y el predicado $p[y]$ en el efecto. Entonces si la variable x e y son instanciadas al mismo objeto, ambos predicados se convierten en el mismo fact. Por lo tanto, puede ocurrir en el splitting que este sea eliminado o agregado previo a su evaluación como precondición, haciendo que la aplicabilidad del splitting no se corresponda con la acción original.

Para asegurar la propiedad (3) vamos a diseñar un framework que caracteriza los posibles splittings atómicamente bien ordenados, mientras que para poder asegurar las propiedades (1) y (2) es suficiente con implementar un sistema de decoración utilizando nuevos predicados en los sub-esquemas que componen un splitting. Comenzaremos por definir en detalle el framework para la propiedad (3) que resulta ser, quizás, lo más complicado de la técnica de splitting. Luego introduciremos el sistema de decoración para de esta manera lograr alcanzar las tres propiedades mencionadas.

4.2. Splittings Válidos

Para la formulación de nuestro framework resulta útil introducir la noción de átomos anotados con la finalidad de identificar el lugar donde ocurre un

predicado en el esquema de acción. Es decir, si la ocurrencia se trata de una precondición, un efecto positivo (en lista *add*) o negativo (en lista *del*).

Definición 28 (Atomos Anotados) *Sea $a[X]$ un esquema de acción STRIPS y $p[Y]$ un átomo que ocurren en $a[X]$. Entonces el* átomo anotado *correspondiente a $p[Y]$, denotado por $Ann(p[Y])$, está definido de la siguiente manera:*

$$Ann(p[Y]) = \begin{cases} p_{pre}[Y] & \text{si } p[Y] \in pre(a[X]) \\ p_{del}[Y] & \text{si } p[Y] \in del(a[X]) \\ p_{add}[Y] & \text{si } p[Y] \in add(a[X]) \end{cases} \tag{4.1}$$

Además, denotaremos con $AnnAt(a[X])$ al conjunto de todos los átomos anotados de $a[X]$.

De la definición anterior, notar que un átomo anotado es simplemente un predicado que cuenta con una anotación que indica en qué lugar del esquema original ocurre ese predicado. Por lo tanto, para cada átomo $p[X]$ en la precondición de un esquema, tenemos un átomo anotado $p_{pre}[X]$. Análogamente, para los predicados de la lista *del* y *add*. Los átomos anotados son convenientes, ya que, los sub-esquemas de un splitting contienen una parte del esquema original, es decir, un subconjunto de $AnnAt(a[X])$. Por lo tanto, vamos a identificar esquemas (y sub-esquemas) con su conjunto de átomos anotados. De hecho, esto es asumido en nuestra definición de splitting. De esta manera, el esquema $Move(x, y, z)$ es representado a través del conjunto de átomos anotados $\{on_{pre}(x, y), clear_{pre}(x), clear_{pre}(z), on_{del}(x, y), clear_{del}(z), on_{add}(x, z), clear_{add}(y)\}$. Así, un splitting del esquema $Move(x, y, z)$ es simplemente una partición de este conjunto, donde cada parte se corresponde con un sub-esquema del splitting.

Definición 29 (Función de Splitting) *Sea $A[Z]$ un conjunto de esquemas STRIPS y sea $\mathcal{A}[Z]$ el conjunto de todos los posibles esquemas sobre $At(A[Z])$. Una función de splitting para $A[Z]$ es una función $\sigma : A[Z] \mapsto \mathcal{P}(\mathcal{A}[Z])$, tal que, siempre que $\sigma(a[X]) = \{a_1[X_1], \ldots, a_n[X_n]\}$, entonces $a_1[X_1], \ldots, a_k[X_n]$ es una partición de $a[X]$, i. e., $a_i[X_i] \cap a_i[X_i] = \emptyset$ para $i \neq j$ y $\bigcup_i a_i[X_i] = a[X]$.*

Notar que el splitting de un esquema de acción está definido como un conjunto de sub-esquemas, por lo tanto, todavía no hay una noción de orden entre ellos. Sin embargo, para asegurar la propiedad (3) de la sección 4.1 vamos a definir una orden parcial sobre los átomos anotados de un esquema que implica una restricción en el orden de ejecución de los sub-esquemas que componen el splitting. Un splitting $\{a_1[X1] \ldots a_n[X_n]\}$ será válido si existe

una forma de ordenar los sub-esquemas, tal que, dicho orden respeta el orden parcial entre los átomos anotados del esquema. Siempre que $p_{pre}[X] \in a_i[X_i]$ y $p_e[Y] \in a_j[X_j]$ con $e = \{del, add\}$, entonces $a_i[X_i]$ debe ser ejecutado en el splitting antes que $a_j[X_j]$. Respetar el orden parcial asegura que los átomos que ocurren simultáneamente en la precondición y en el efecto de un acción, sean siempre ejecutados primero como precondición y luego como efecto. De igual manera, si un átomo ocurre en la lista *del* y *add*, entonces la ocurrencia en *del* debe ser ejecutada antes que su ocurrencia en *add* para preservar la semántica, ya definida, donde los efectos positivos prevalecen por sobre los efectos negativos cuando un mismo efecto ocurre en ambas listas.

Definición 30 (Orden Parcial $\leq$) *Sea $a[X]$ un esquema de acción STRIPS y sean $p_e[Y]$, $p_{e'}[Y']$ átomos anotados de $a[X]$. Entonces definimos $p_e[Y] \leq p_{e'}[Y']$ sii $e = pre$ y $e' = del$; ó $e = pre$ y $e' = add$; ó $e = del$ y $e' = add$.*

Un interpretación más intuitiva de la relación $p_e[Y] \leq p_{e'}[Y']$ es que indica que $p_e[Y]$ debe ocurrir en el splitting antes que $p_{e'}[Y']$. Extendemos esta relación entre átomos anotados a conjuntos de átomos anotados en la forma natural. Sean S y S' conjuntos de átomos anotados y distintos, entonces definimos $S \leq S'$ sii existen $p_e[Y] \in S$ y $p_{e'}[Y'] \in S'$, tal que, $p_e[Y] \leq p_{e'}[Y']$. Recordar que habíamos mencionado que los esquemas y sub-esquemas son considerados conjuntos de átomos anotados. Por lo tanto, la definición de $\leq$ se encuentra bien definida para estos.

Definición 31 (Secuencialización) *Sea $a[X]$ un esquema de acción STRIPS y sea $\sigma(a[X]) = \{a_1[X_1], \ldots, a_n[X_n]\}$ un splitting de $a[X]$, entonces una* secuencialización *del splitting es cualquier orden lineal $a_{i_1}[X_{i_1}], \ldots, a_{i_n}[X_{i_n}]$. Diremos que una secuencialización es* correcta *si respeta el orden parcial $\leq$, i. e., siempre que $a_{i_j}[X_{i_j}] \leq a_{i_{j'}}[X_{i_{j'}}]$, entonces $j \leq j'$.*

El concepto de secuencialización impone un orden de ejecución entre los sub-esquemas del splitting. Aquellas secuencializaciones, que no violan la relación $\leq$ entre los sub-esquemas, aseguran que el splitting resultante sea atómicamente bien ordenado. Para terminar de comprender la importancia de una correcta secuencialización, consideremos un simple ejemplo donde tenemos un esquema $a[x, y] = \{p_{del}(x), p_{add}(y)\}$ y el splitting $a_1[x] = \{p_{del}(x)\}$, $a_2[y] = \{p_{add}(y)\}$ que separa cada átomo en un sub-esquema distinto. Notar que al finalizar la ejecución del esquema original siempre vale $p(y)$. Sin embargo, si x e y se instancian al mismo objeto y el sub-esquema $a_2[y]$ es ordenado antes que $a_1[x]$ en el splitting, entonces al finalizar la ejecución del splitting el fact $p(y)$ no vale. Provocando que el resultado de la aplicación del

splitting sea distinto al de la acción original. Esta inconsistencia se produjo como consecuencia de que la secuencialización $a_2[y], a_1[x]$, no es correcta. De hecho, la única secuencialización correcta posible es $a_1[x], a_2[y]$, la cual, evita claramente la inconsistencia recién descripta. En definitiva, la noción de secuencialización correcta nos permite caracterizar aquellos splittings que aseguran la propiedad (3).

Definición 32 (Splitting Válido) *Sea $a[X]$ un esquema de acción STRIPS y sea $\sigma(a[X])$ un splitting de $a[X]$, entonces $\sigma(a[X])$ es un splitting válido sii existe una secuencialización correcta de $\sigma(a[X])$.*

Observar que la relación $\leq$ sobre los sub-esquemas de un splitting define un grafo. Este grafo resulta muy útil para decidir si un splitting es válido y obtener una secuencialización correcta del mismo.

Definición 33 (Grafo Cociente) *Sea $a[X]$ un esquema de acción STRIPS y sea $\sigma(a[X])$ un splitting de $a[X]$, entonces el* grafo cociente *de $\sigma(a[X])$ es el grafo dirigido $(\sigma(a[X]), \leq)$.*

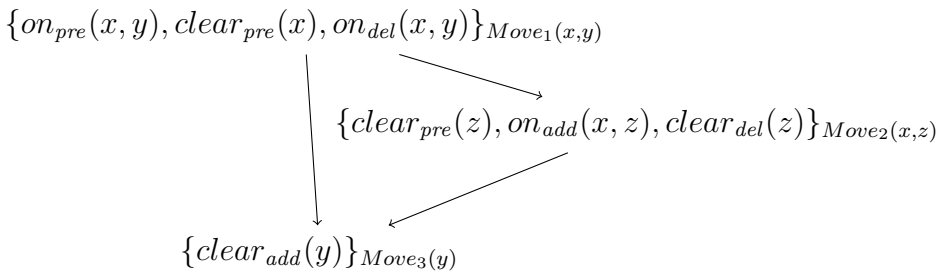
A continuación, mostramos el grafo cociente del splitting tentativo del esquema $Move(x, y, z)$ propuesto en el ejemplo 1:

$$\{on_{pre}(x,y), clear_{pre}(x), clear_{add}(y), on_{del}(x,y)\}_{Move_1(x,y)}$$

$$\updownarrow$$

$$\{clear_{pre}(z), on_{add}(x,z), clear_{del}(z)\}_{Move_2(x,z)}$$

Notar que el grafo presenta un ciclo producto de que el átomo $on_{pre}(x, y)$ debe ser ejecutado antes que el átomo $on_{add}(x, z)$. Esta restricción es representada por la arista en sentido hacia arriba. El átomo $clear_{del}(z)$ debe ser ejecutado antes que $clear_{add}(y)$ y esta restricción es indicado por la arista en sentido hacia abajo. De esta manera, se genera un ciclo que no permite que el splitting posea una secuencialización correcta. Esto demuestra que el splitting $Move_1(x, y), Move_2(x, z)$ inicialmente propuesto pare el esquema $Move(x, y, z)$ no es válido. Una manera de eliminar tal ciclo es crear un tercer sub-esquema $Move_3(y)$, que sólo contenga el átomo $clear_{add}(y)$. Así, generamos un nuevo splitting con el siguiente grafo cociente:



Este último grafo cociente no presenta ciclos, Por lo tanto, el splitting correspondiente tiene una secuencialización correcta que asegura la propiedad (3). Observar que en este nuevo splitting si instanciamos las variables y y z al mismo bloque, entonces la ejecución del splitting falla en la sub-acción $Move_2$ cuando la precondición $clear(z)$ es ejecutada debido a que ese fact ya no es agregado previamente por la sub-acción $Move_1$. En conclusión, el grafo cociente de un splitting resulta muy útil para determinar si un splitting es válido. Y en caso de serlo, además obtener una secuencialización correcta del mismo.

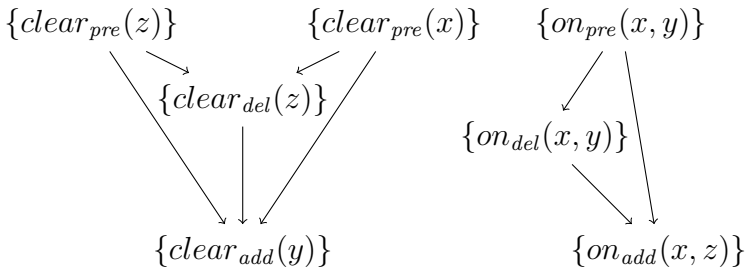
Proposición 9 *Sea $a[X]$ un esquema de acción STRIPS, y sea $\sigma(a[X])$ un splitting de $a[X]$. Entonces $\sigma(a[X])$ es válido sii su grafo cociente no tiene ciclos.*

Demo 10 *Directo de la definición de orden parcial $\leq$ y por construcción de grafo cociente de $\sigma(a[X])$.*

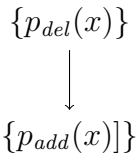
Entonces, gracias a la proposición anterior contamos con un mecanismo muy simple para verificar si un splitting dado es válido. Pero surge la pregunta sobre cómo construir un splitting válido dado un esquema de acción cualquiera. Para responder esta pregunta, primero observemos que para todo esquema de acción $a[X]$ existen dos splittings triviales extremadamente opuestos. Uno de ellos separa cada átomo anotado del esquema en un sub-esquema distinto y el otro coloca todos los átomos anotados en un único sub-esquema.

Definición 34 *Sea $a[X]$ un esquema de acción STRIPS. Entonces el splitting trivial de $a[X]$, denotado por Trivial$(a[X])$, es el conjunto $\{a[X]\}$ que asignada a cada átomo anotado de $a[X]$ al mismo y único sub-esquema. Y el splitting atómico de $a[X]$, denotado por Atomic$(a[X])$, es el conjunto $\{\{p_e[Y]\} \mid p_e[Y] \in a[X]\}$ que asigna a cada átomo anotado de $a[X]$ un sub-esquema distinto.*

Es suficiente con que el esquema $a[X]$ contenga más de un predicado para que $Atomic(a[X])$ contenga más de un sub-esquema, y por ende, ser distinto a $Trivial(a[X])$. En particular, para el caso del esquema $Move(x, y, z)$, el splitting atómico consiste en 7 sub-esquemas. Cada uno de ellos contiene solamente uno de los siete átomos anotados del esquema original. Su grafo cociente se muestra a continuación:



Reconsiderando el ejemplo donde teníamos un esquema de acción $a[x, y] = \{p_{del}[x], p_{add}[y]\}$. El splitting que separaba ambos átomos anotados resulta ser el splitting atómico del esquema. Su grafo cociente es:



Este grafo ordena el sub-esquema que contiene a $p_{del}[x]$ antes que el sub-esquema que contiene a $p_{add}[y]$. Por lo tanto, la única secuencialización correcta posible es la que hace valer a $p(x)$ al finalizar la ejecución del splitting. Esto se corresponde exactamente a lo que sucede al ejecutar la acción original $a(x, y)$. En definitiva, es sencillo demostrar que el grafo cociente de un splitting atómico es acíclico, lo que implica su validez.

Proposición 10 (Validez Atómica y Trivial) *Sea $a[X]$ un esquema de acción STRIPS, entonces $Atomic(a[X])$ y $Trivial(a[X])$ son splittings válidos.*

Demo 11 *El grafo cociente de $Trivial(a[X])$ tiene un único vértice, y al ser irreflexivo por su definición, entonces no puede tener ciclos. El grafo cociente de $Atomic(a[X])$ tiene un vértice por cada átomo anotado de $a[X]$; observar que el peor caso posible se presenta cuando un mismo predicado ocurre simultáneamente en la precondición, en la lista del y add de $a[X]$. Sin embargo, este caso no genera ningún ciclo, porque la ocurrencia del predicado en la precondición se ordena antes que su ocurrencia en del y add; y su ocurrencia en del es ordenada antes que su ocurrencia en add.*

A partir de estos dos splitting válidos podemos hallar splitting válidos más interesantes. Esta afirmación se basa en la observación de que el splitting atómico y trivial son los casos extremos de una jerarquía de splittings válidos. El splitting atómico es aquel que más particiona el esquema original (máxima granularidad) mientras que el splitting trivial es el que menos particiona (mínima granularidad). Por consiguiente, la idea consiste en hallar un splitting válido con un nivel de granularidad intermedio. Diremos que $\sigma(a[X])$ tiene mayor *granularidad* que $\sigma'(a[X])$ si $\sigma(a[X]) \neq \sigma'(a[X])$ y para cada $a_i[X_i] \in \sigma(a[X])$, existe $a'_j[X'_j] \in \sigma'(a[X])$, tal que, $a_i[X_i] \subseteq a'_j[X'_j]$. Entonces vamos a explorar esta jerarquía de granularidad. Partiendo del splitting atómico que posee el máximo nivel de granularidad, fusionando en cada paso dos sub-esquemas del splitting para obtener un nuevo splitting con un menor nivel de granularidad. Repitiendo este proceso sucesivamente hasta converger al splitting trivial, el cual, tiene el mínimo nivel de granularidad. Obviamente, en cada paso, hay que tener la precaución de que la fusión (merge) de dos sub-esquemas no genere un ciclo en el grafo cociente del splitting resultante.

Definición 35 *Sea $a[X]$ un esquema de acción STRIPS y $\sigma(a[X]) = \{a_1[X_1], \ldots, a_n[X_n]\}$ un splitting de $a[X]$. Denotamos con $\leq_*$ la clausura transitiva del grafo cociente de $\sigma(a[X])$. Entonces diremos que $a_i[X_i]$ y $a_j[X_j]$ son sub-esquemas* mergeables *de $\sigma(a[X])$ sii no existe $k \neq i, j$ tal que $a_i[X_i] \leq_* a_k[X_k] \leq_* a_j[X_j]$ o $a_j[X_j] \leq_* a_k[X_k] \leq_* a_i[X_i]$. El splitting resultante de fusionar $a_i[X_i]$ y $a_j[X_j]$ es aquel que se obtiene de reemplazar los sub-esquemas $a_i[X_i]$ y $a_j[X_j]$ por el sub-esquema $a_i[X_i] \cup a_j[X_j]$ en $\sigma(a[X])$ y lo denotaremos con $\sigma(a[X])^{i,j}$.*

La noción de sub-esquemas mergeables es importante. Este se convierte en un constructor de splitting válidos: si fusionamos dos sub-esquemas mergeables de un splitting válido, entonces obtenemos siempre un nuevo splitting válido.

Proposición 11 *Sea $a[X]$ un esquema de acción y sea $\sigma(a[X]) = \{a_1[X_1], \ldots, a_n[X_n]\}$ un splitting válido de $a[X]$. Si $a_i[X_i]$ y $a_j[X_j]$ son sub-esquemas mergeables de $\sigma(a[X])$, entonces el splitting $\sigma(a[X])^{i,j}$ es válido.*

Demo 12 *Por el absurdo. Supongamos que el merge de $a_i[X_i]$ y $a_j[X_j]$ introduce un ciclo y sea $a_k[X_k]$ un nodo del ciclo diferente de $a_i[X_i] \cup a_j[X_j]$. Entonces $a_i[X_i] \cup a_j[X_j] \leq_* a_k[X_k]$ y $a_k[X_k] \leq_* a_i[X_i] \cup a_j[X_j]$. Pero entonces, $a_i[X_i] \leq_* a_k[X_k]$ ó $a_j[X_j] \leq_* a_k[X_k]$ y $a_k[X_k] \leq_* a_i[X_i]$ ó $a_k[X_k] \leq_* a_j[X_j]$ en el grafo cociente de $\sigma(a[X])$. Esto último, genera cuatro casos posibles.*

Caso 1: $a_i[X_i] \leq_* a_k[X_k]$ *y* $a_k[X_k] \leq_* a_i[X_i]$*, absurdo pues el grafo cociente de* $\sigma(a[X])$ *no tiene ciclos. Lo mismo sucede para el caso* $a_i[X_i] \leq_* a_k[X_k]$ *y* $a_k[X_k] \leq_* a_j[X_j]$*. Caso 2:* $a_i[X_i] \leq_* a_k[X_k]$ *y* $a_k[X_k] \leq_* a_j[X_j]$*, absurdo, pues* $a_i[X_i]$ *y* $a_j[X_j]$ *son mergeables. Lo mismo sucede para el caso* $a_j[X_j] \leq_* a_k[X_k]$ *y* $a_k[X_k] \leq_* a_i[X_i]$*.*

Finalmente, estamos en condiciones de caracterizar el espacio de todos los splitting válidos de un esquema de acción.

Teorema 5 *Sea* $a[X]$ *un esquema de acción STRIPS. Entonces cualquier splitting que se obtiene, a partir de* $Atomic(a[X])$*, iterativamente fusionando sub-esquemas mergeables es válido. Viceversa, cualquier splitting válido de* $a[X]$ *puede ser construído de esta manera.*

Demo 13 *La ida es consecuencia directa de la proposición 10 y 11. Veamos la vuelta. Sea* $\sigma(a[X]) = \{a_1[X_1], \ldots, a_n[X_n]\}$ *un splitting válido. Hacemos inducción en n. Caso base: trivial, pues si* $n = |Atomic(a[X])|$ *entonces* $\sigma(a[X]) = Atomic(a[X])$*. Caso inductivo: supongamos* $\sigma(a[X]) = \{a_1[X_1], \ldots, a_{n-1}[X_{n-1}]\}$ *un splitting válido de* $a[X]$*, por ende, el grafo cociente de* $\sigma(a[X])$ *no tiene ciclos. Por lo tanto, debe existir un sub-esquema* $a_i[X_i]$ *tal que tiene una partición no trivial* $\{a_{i_1}[X_{i_1}], a_{i_2}[X_{i_2}]\}$ *que no genera un ciclo en el grafo cociente del splitting* $\sigma'(a[X]) = \{a_1[X_1], \ldots, a_{i_1}[X_{i_1}], a_{i_2}[X_{i_2}], \ldots, a_{n-1}[X_{n-1}]\}$*. Luego,* $\sigma'(a[X])$ *es un splitting válido de* $a[X]$ *con n sub-esquemas. Por hipótesis inductiva,* $\sigma'(a[X])$ *se puede construir a partir de* $Atomic(a[X])$ *iterativamente fusionando sub-esquemas mergeables. Y como* $a_{i_1}[X_{i_1}]$ *y* $a_{i_2}[X_{i_2}]$ *son sub-esquemas mergeables en* $\sigma'(a[X])$*, entonces* $\sigma(a[X])$ *se puede obtener a partir de* $\sigma'(a[X])$ *fusionado estos dos sub-esquemas. Por lo tanto,* $\sigma(a[X])$ *se puede obtener a partir de* $Atomic(a[X])$ *iterativamente fusionando sub-esquemas mergeables.*

Reconsideremos el splitting válido del esquema $Move(x, y, z)$ compuesto por los tres sub-esquemas $Move_1(x, y)$, $Move_2(x, z)$, $Move_3(y)$. Este splitting puede ser obtenido a partir del splitting atómico de $Move(x, y, z)$ fusionando iterativamente el sub-esquema $\{clear_{pre}(x)\}$ con $\{on_{pre}(x, y)\}$ y luego con $\{on_{del}(x, y)\}$; y por otro lado, fusionando el sub-esquema $\{clear_{pre}(z)\}$ con $\{on_{add}(x, z)\}$ y luego con $\{clear_{del}(z)\}$.

En conclusión, estos resultados sugieren que podemos explorar el espacio de splitting válidos de un esquema partiendo del splitting atómico y fusionando sub-esquemas mergeables, hasta encontrar un splitting que consideramos que posee el nivel de granularidad adecuado. En la sección 4.4 proponemos que el splitting con un nivel de granularidad adecuado es aquel que posee

un buen trade-off entre la cantidad de sub-esquemas (lo que genera un aumento en la distancia a la meta) y la reducción en el tamaño de la interfaz (lo que genera una disminución en la cantidad de acciones proposicionales que produce el splitting con respecto al esquema original). Recordar que el concepto de splitting válido sólo asegura la propiedad (3) de la sección 4.1. Aún resta asegurar la propiedad (1) *instanciación consistente* y la propiedad (2) *ejecución en bloque*. En la próxima sección describimos en detalle cómo asegurar estas dos últimas propiedades.

4.3. Decorando un Splitting Válido

Para asegurar las propiedes (1) y (2) de la sección 4.1 introducimos predicados nuevos para implementar un sistema de token entre los sub-esquemas de un splitting. Vamos a formalizar esta idea en términos de una pequeña modificación (decoración) de la función de splitting.

Definición 36 *Sea $A[Z]$ un conjunto de esquemas de acción y σ una función de splitting para $A[Z]$, tal que, $\sigma(a[X])$ es válido para todo $a[X] \in A[Z]$. Definimos la* función de splitting decorada $\tilde{\sigma}$*, tal que, para todo $a[X]$, selecciona una secuencialización correcta $a_1[X_1], \ldots, a_n[X_n]$ de $\sigma(a[X])$ y es decorada de la siguiente manera:*

(I) ***Token none.*** *Si $n > 1$, entonces $\tilde{\sigma}(a[X])$ agrega el átomo* ***none*** *a $pre(a_1[X_1])$, $del(a_1[X_1])$ y $add(a_n[X_n])$. Si $n = 1$, entonces $\tilde{\sigma}(a[X])$ agrega solamante el átomo* ***none*** *a $pre(a_1[X_1])$.*

(II) ***Token do.*** *Asumamos una función biyectiva $id : A[Z] \mapsto \{1, \ldots, |A[Z]|\}$. Si $n > 1$, entonces $\tilde{\sigma}(a[X])$ agrega los átomos $\mathbf{do}_2^{id(a[X])}$ a $add(a_1[X_1])$; $\mathbf{do}_j^{id(a[X])}$ a $pre(a_j[X_j])$ y $del(a_j[X_j])$; and $\mathbf{do}_{j+1}^{id(a[X])}$ a $add(a_j[X_j])$, para todo $1 < j < n$; $\mathbf{do}_n^{id(a[X])}$ a $pre(a_n[X_n])$ y $del(a_n[X_n])$. Si $n = 1$, entonces el átomo* ***do*** *no es utilizado.*

(III) ***Token var***. *Asumamos una función biyectiva $id : X \mapsto \{1, \ldots, |X|\}$. Si $x \in X_i \cap X_j$ con $i \neq j$, entonces $\tilde{\sigma}(a[X])$ agrega el átomo $\mathbf{var}^{id(x)}(x)$ a $add(a_{jmin}[X_{jmin}])$ donde $jmin$ es el menor j tal que $x \in X_j$; a $pre(a_j[X_j])$ para cada $j > jmin$ tal que $x \in X_j$; a $del(a_{jmax}[X_{jmax}])$ donde $jmax$ es el mayor j tal que $x \in X_j$.*

Con $\tilde{\sigma}(A[Z]) = \bigcup_{a[X] \in A[Z]} \tilde{\sigma}(a[X])$ vamos a denotar al conjunto de sub-esquemas obtenidos al aplicar $\tilde{\sigma}$ a los esquemas de $A[Z]$. Además, nos referiremos a $\tilde{\sigma}(A[Z])$ como el splitting de $A[Z]$ obtenido vía σ.

El token *none* y *do* conjuntamente aseguran que las sub-acciones de un splitting sólo puedan ser ejecutadas en bloque, es decir, que sean ejecutadas todas y consecutivamente. El token *none* logra que ningún otro splitting puede comenzar a ser ejecutado, ya que, una vez que ejecutamos la primer sub-acción de un splitting el token *none* es liberado recién en la última sub-acción y todo splitting necesita de este token para comenzar su ejecución. El token *do* asegura que la ejecución de un splitting sea de acuerdo a la secuencialización correcta seleccionada. Al estar el token indexado por un *id* que indica a que esquema de acción pertence el splitting, esto asegura que ninguna sub-acción perteneciente a otra acción pueda ser ejecutada. Además, garantiza que cada sub-acción sea ejecutada una única vez durante el tiempo que dure la ejecución del splitting. El token *var* asegura la instanciación consistente de las variables comunes entre los sub-esquemas que componen un mismo splitting. Este token está indexado por un *id* de variable, ya que, de otra manera, los roles de dos variables compartidas podrían intercambiarse sutilmente. Por ejemplo, supongamos que las variables x e y son instanciadas inicialmente con los objetos o_1 y o_2 por algún sub-esquema del splitting. Y supongamos que existe un sub-esquema posterior en el splitting que contiene a x e y en su interfaz. Entonces los tokens sin *id* $var(o_1)$ y $var(o_2)$ podrían ser utilizados para permitir instanciar x con o_2 e y con o_1. Es decir, al revés de como fueron esas variables instanciadas en la primer sub-acción. Claramente, esto no puede ocurrir si agregamos al token un sub-índice que indica qué variables del esquema original estamos instanciando durante el splitting. Para terminar de comprender el sistema de tokens descripto, en el ejemplo 2, mostramos la decoración del splitting válido de tres sub-acciones del esquema $Move(x, y, z)$.

Ejemplo 2 *Implementación del sistema de decoración del splitting válido de tres sub-acciones del esquema* $Move(x, y, z)$. *Sea* $id(Move(x, y, z)) = 1$, $id(x) = 1$, $id(y) = 2$. *Entonces, el splitting decorado* $\tilde{\sigma}(Move(x, y, z))$ *está definido de la siguiente manera:*

$\boldsymbol{Move}_1(x, y)$
$pre : \{on(x, y), clear(x), \boldsymbol{none}\}$
$del : \{on(x, y), \boldsymbol{none}\}$
$add : \{\boldsymbol{do}_2^1, \boldsymbol{var}^1(x), \boldsymbol{var}^2(y)\}$

$\boldsymbol{Move}_2(x, z)$
$pre : \{clear(z), \boldsymbol{do}_2^1, \boldsymbol{var}^1(x)\}$
$del : \{on(x, y), \boldsymbol{none}\}$
$add : \{on(x, z), \boldsymbol{do}_3^1\}$

$\boldsymbol{Move}_3(y)$
$pre : \{\boldsymbol{do}_3^1, \boldsymbol{var}^2(y)\}$
$del : \{\boldsymbol{do}_3^1, \boldsymbol{var}^2(y)\}$
$add : \{clear(y), \boldsymbol{none}\}$

Notar que los sub-esquemas solo pueden ser ejecutados en el orden establecido por el sistema de tokens. El primer sub-esquema del splitting toma el token *none* y este es liberado por el último sub-esquema. Las variables x e y tiene que ser instanciadas consistentemente. $Move_2(x, z)$ tiene que obtener el token $var^1(x)$ desde la sub-acción $Move_1(x, y)$ y $Move_3(y)$ tiene que obtener el token $var^2(y)$ también desde la sub-acción $Move_1(x, y)$. Notar que el sistema de tokens asegura la secuencialización correcta del splitting. De esta manera, logramos alcanzar las tres propiedades que debe satisfacer un splitting para asegurar que los planes no resulten afectados por esta operación. Por último, para que cualquier tarea de planning producto de un splitting preserve los planes en una relación uno a uno, resta incluir el átomo *none* en el estado inicial y en la meta de la tarea:

Teorema 6 *Sea $A[Z]$ un conjunto de esquemas de acción y σ una función de splitting para $A[Z]$, tal que, $\sigma(a[X])$ es válido para todo $a[X] \in A[Z]$. Sea A y $A^{\tilde{\sigma}}$ el conjunto de acciones proposicionales obtenidas mediante el proceso de grounding cartesiano sobre $A[Z]$ y $\tilde{\sigma}(A[Z])$, respectivamente. Entonces los planes de $\Pi = (At(A), A, s_0, g)$ con $s_0, g \subseteq At(A)$, están en una correspondencia uno a uno con los planes de $\Pi^{\tilde{\sigma}} = (At(A^{\tilde{\sigma}}), A^{\tilde{\sigma}}, s_0 \cup \{\boldsymbol{none}\}, g \cup \{\boldsymbol{none}\})$.*

Demo 14 *Cualquier plan de Π puede ser transformado a su correspondiente plan de $\Pi^{\tilde{\sigma}}$ en foma directa: simplemente reemplazando cada acción instanciada original con su splitting instanciado. Y viceversa, cualquier plan de $\Pi^{\tilde{\sigma}}$ puede ser transformado a su correspondiente plan de Π en forma inversa gracias a que $\tilde{\sigma}$ cumple con las propiedades de instanciación consistente, ejecución en bloque y secuencialización correcta.*

La importancia de splitting como técnica de optimización radica en la cantidad de acciones proposicionales de $A^{\tilde{\sigma}}$ que debe ser mucho menor a la cantidad acciones proposicionales en A. Esto es producto de que el splitting reduce el tamaño de las interfaces del dominio original. Además, esto se logra sin afectar los planes de la tarea original, es decir, podemos recuperar un plan de la tarea original a través de un plan de la tarea spliteada. Entre las desventajas de splitting podemos mencionar que aumenta la distancia a la meta (lo cual puede repercutir desfavorablemente en la etapa de búsqueda) y que *no* preserva optimalidad. Un plan óptimo de Π no necesariamente se corresponde con un plan óptimo de $\Pi^{\tilde{\sigma}}$, y viceversa. Esto se debe a que los esquemas que tienen un splitting de mayor longitud (más sub-esquemas) son penalizados con respectos a aquellos que poseen un splitting de menor longitud. Aunque esto puede ser subsanado asignando un costo uniforme a cada sub-esquema $1/|\tilde{\sigma}(a[X])|$ en un setup que incluya costos de acción.

En la próxima sección estudiamos como automatizar algorítmicamente la obtención de un "buen" splitting válido y su complejidad computacional utilizando los conceptos vistos hasta este momento.

4.4. Computando un Splitting

En esta sección vamos a estudiar como automatizar la operación de splitting, qué es y cómo computar un buen splitting. En la sección anterior, vimos que existe una jerarquía de splittings válidos entre el splitting atómico (todos los átomos anotados separados en un sub-esquema distinto) y el splitting trivial (todos los átomos anotados en un mismo y único sub-esquema). A medida que nos acercamos al splitting trivial en dicha jerarquía, los splittings se caracterizan por tener una menor cantidad de sub-esquemas (y por ende, los planes tienden a ser más cortos). Pero con la contraparte de no modificar sustancialmente el tamaño de las interfaces del esquema original. En consecuencia no tendríamos una reducción significativa de la cantidad de acciones proposicionales. Por el otro lado, a medida que nos acercamos al splitting atómico, los splittings tienden a tener más sub-esquemas (y por ende, tenemos planes más largos). Pero con un una gran reducción en el tamaño de la interfaz de los esquemas, lo cual, puede provocar una reducción muy significativa en la cantidad de acciones proposicionales. Vamos a capturar este trade-off en términos de dos variables. Por un lado, el tamaño del splitting, definido como la cantidad de sub-esquemas del mismo, es decir, $\mathit{SplitSize}(\sigma(a[X])) = |\sigma(a[X])|$. Por el otro lado, el tamaño de la interfaz del splitting, definido como el tamaño de la interfaz más grande de los sub-esquemas que componen el mismo, es decir, $\mathit{IntSize}(\sigma(a[X])) = \text{máx}_{a_i[X_i] \in \sigma(a[X])} |X_i|$. Observar que el largo de los planes se incrementa linealmente en $\mathit{SplitSize}(\sigma(a[X]))$. Obviamente, cuando la acción en cuestión ocurre en un plan. Mientras que el número de acciones proposicionales generadas por el splitting decrece exponencialmente en $|X| - \mathit{IntSize}(\sigma(a[X]))$.

4.4.1. Su Complejidad Computacional

Con respecto a las dos funciones del trade-off propuesto, observar que el splitting trivial es óptimo en el valor de la función *SplitSize*(), pero tiene el máximo valor posible en la función *IntSize*(). Por el otro lado, en el caso del splitting atómico es óptimo en el valor de *IntSize*(), pero tiene el peor valor para la función *SplitSize*(). Entonces, queremos hallar un splitting válido con el mejor trade-off posible entre estas dos funciones. Desafortunadamente,

encontrar un splitting válido con el trade-off óptimo es difícil de computar.

Teorema 7 *Sea* ***Splitting-Optimization*** *el problema de decidir si existe, dado un esquema de acción $a[X]$ y naturales K, N, un splitting válido $\sigma(a[X])$, tal que, $SplitSize(\sigma(a[X])) \leq K$ y $IntSize(\sigma(a[X])) \leq N$. Entonces Splitting-Optimization es NP-completo.*

Demo 15 *Veamos pertenencia a NP. Verificar si un conjunto de sub-esquemas es un splitting, es decir, chequear si los sub-esquemas son una partición del esquema original claramente es polinomial. Además, verificar su validez, y si su cantidad de sub-esquemas y su tamaño de interfaz son menores a K y N respectivamente, también es polinomial, por lo tanto, Splitting-Optimization $\in NP$. Veamos hardness: vía reducción polinomial del problema BinPacking. Dado una instancia $\langle\{p_1, \ldots p_n\}, T, B\rangle$ del problema BinPacking con p_i un item, T bins y B la capacidad de los bins, entonces podemos construir el esquema de acción $a[X]$ con $pre(a[X]) = \{p_i(x_1^i, \ldots, x_{p_i}^i) \mid i = 1 \ldots n\}$, $del(a[X]) = add(a[X]) = \emptyset$ y tomar $K = T$ y $N = B$. Esta codificación es polinomial con respecto a $\langle\{p_1, \ldots p_n\}, T, B\rangle$. Y por último, es directo ver que $\langle p_1, \ldots p_n, T, B\rangle$ tiene solución sii $a[X]$ tiene un splitting válido σ, tal que, $SplitSize(\sigma) \leq T$ y $IntSize(\sigma) \leq B$.*

El resultado anterior indica que no es posible encontrar un algoritmo eficiente, es decir, polinomial que compute un splitting con el trade-off óptimo, salvo que $P = NP$. En consecuencia, vamos a implementar un algoritmo que aproxima la solución óptima. En otras palabras, en la práctica vamos a computar un splitting válido con un buen trade-off (mínimo local), en lugar de aquel con el mejor trade-off posible (mínimo global).

4.4.2. Aproximación por Hill Climbing

Matemáticamente vamos a describir el trade-off entre las dos variables ya mencionadas mediante una suma ponderada de las mismas, donde cada una es normalizada en el intervalo $[0, 1]$. Entonces, el *tamaño normalizado del splitting* es: [2]

$$NormSplitSize(\sigma) = \frac{SplitSize(\sigma)}{SplitSize(Atomic(a[X]))}$$

.

[2]Por simplicidad, denotaremos al splitting de un esquema de acción $\sigma(a[X])$ simplemente con σ.

Y *el tamaño normalizado de la interfaz del splitting* es:

$$NormIntSize(\sigma) = \frac{IntSize(\sigma)}{IntSize(\,Trivial(a[X]))}$$

Entonces nuestro problema de optimización consiste en encontrar un splitting válido σ, tal que, minimiza la siguiente ecuación:

$$TradeOff\,(\sigma) = \gamma NormSplitSize(\sigma) + (1-\gamma)NormIntSize(\sigma)$$

donde el parámetro $\gamma \in [0,1]$ controla el trade-off entre ambas variables.

Para esto vamos a realizar una búsqueda *hill-climbing* en la jerarquía de splittings válidos. Comenzando por el splitting atómico, escalando hacia splittings con menor nivel de granularidad, mediante merge de sub-esquemas mergeables, hasta converger al splitting trivial. De todos los splitting visitados durante la búsqueda, elegimos devolver aquel que tuvo un $TradeOff\,(\sigma)$ mínimo. En definitiva, realizamos una búsqueda instanciando el algoritmo Hill Climbing de la siguiente manera:

- **Nodo inicial:** $\sigma = Atomic(a[X])$
- **Función Sucesora:**

  $$Suc(\sigma) = \{\sigma^{i,j} \mid \forall i \neq j \textit{ tal que } a_i[X_i] \textit{ y } a_j[X_j] \textit{ son mergeables en } \sigma\}$$

- **Sucesor a Expandir:** $\sigma^{i,j} \in Suc(\sigma)$ con menor $TradeOff\,(\sigma^{i,j})$
- **Condición de Terminación:** $\sigma = Trivial(a[X])$
- **Retorno:** σ expandido con menor $TradeOff\,(\sigma)$

Hill-Climbing genera iterativamante splittings válidos mediante merge entre pares de sub-esquemas, expandiendo en cada iteración aquel sucesor con mejor trade-off, hasta llegar al splitting trivial. Analicemos los casos extremos. Cuando $\gamma = 1$, todo el peso de la suma ponderada es asignado al tamaño del splitting. Por lo tanto, la búsqueda finaliza retornando $Trivial(a[X])$. Cuando $\gamma = 0$, todo el peso está en el tamaño de la interfaz del splitting. Por lo tanto, la búsqueda finaliza retornando un splitting con valor óptimo en el tamaño de la interfaz, igual al splitting atómico. Pero, tal vez, con una mejora en el tamaño del splitting, o sea, con menos sub-esquemas.

4.4.3. Secuencializaciones Preferibles

Observar que pueden existir varias secuencializaciones correctas para un mismo splitting válido. Por lo tanto, una vez que Hill-Cimbing retorna un splitting válido σ con un buen trade-off, aún resta escoger una secuencialización correcta de σ, si hay más de una. Entonces, surge la pregunta acerca de que si existe una secuencialización correcta más conveniente que otra. Una posible respuesta consiste en la observación de que si el splitting no es aplicable en un estado s, entonces lo más conveniente es que la primera sub-acción del splitting sea no aplicable en s. Es decir, que el planificador detecte la no aplicabilidad de un splitting lo antes posible. Para terminar de comprender esto, supongamos una acción a que no es aplicable en un estado s debido a que el fact p, que pertenece a la precondición de a, no vale en s. Entonces existe una sub-acción a_i que tiene a p en su precondición y es ésta misma sub-acción la que hace que el splitting sea no aplicable en s. Si a_i es la última sub-acción del splitting, entonces el planificador puede llegar a aplicar las $n-1$ sub-accciones primeras del splitting para finalmente no poder aplicar esta última. En cambio, si la sub-acción a_i resulta ser la primera del splitting, entonces el planificador evita aplicar sub-acciones innecesariamente. En conclusión, preferimos aquellas secuencializaciones correctas que colocan al principio del splitting a aquellas sub-acciones con más posibilidades de ser no aplicables. Esto último, claramente, no es fácil de decir en forma general, pero si podemos apoyarnos en la siguiente heurística: una sub-acción a_i tiene más probabilidad de ser no aplicable en un estado arbitrario s mientras más facts tenga en su precondición. Observar que si a_i no tiene precondiciones, entonces es siempre aplicable en cualquier estado. En el otro extremo, si a_i tiene como precondición a todos los facts F de la tarea, entonces a_i es aplicable en un único estado (F). En definitiva, heurísticamente mientras más precondiciones tienen las sub-acciones, más difícil resulta su aplicabilidad. Por lo tanto, decidimos colocar preferentemente estas sub-acciones al principio del splitting. Por supuesto, siempre respetando primero el orden impuesto por la relación $\leq$ definida entre los sub-esquemas del splitting.

En el algoritmo 3 se muestra el procedimiento de secuencialización implementado en nuestra técnica Action Schema Splitting. Este procedimiento utiliza el grafo cociente del splitting válido que se quiere secuencializar. En cada iteración se obtiene el conjunto de aquellos nodos (sub-esquemas) del grafo cociente que no tienen flechas de entrada y se los elimina del grafo cociente. Luego, se ordena este conjunto en forma decreciente con respecto a la cantidad de facts en sus precondiciones para luego ser agregados a la secuencialización en dicho orden. Este proceso continua hasta que el grafo cociente no tiene más nodos. Claramente, al final de este procedimiento tenemos una

secuencialización del splitting que respeta el orden impuesto por la relación $\rightarrow$ y con los sub-esquemas con más precondiciones ordenados al principio de la secuencialización.

Algorithm 3: Algoritmo de Secuencialización.

Entrada: splitting σ

```
1: g := QuotientGraph(σ)
2: seq := ∅
3: while ¬(g.isEmpty()) do
4:    A_k := g.getNoIncomingEdgeNodes()
5:    g.deleteNodes(A_k)
6:    A_k.sortByPrecondition()
7:    seq.Append(A_k)
8: end while
9: return seq
```

Una pequeña modificación al algoritmo anterior y que es actualmente implementada en nuestro algoritmo de splitting es considerar a los predicados estáticos que ocurren en la precondición de un esquema como más relevante que el resto de los predicados. Esto se logra modificando levemente el método $sortByPrecondition()$ de la línea 6 del algoritmo ya mostrado, de forma tal, que ordene los sub-esquemas del splitting de acuerdo a la cantidad de predicados estáticos en la precondición y en caso de igualdad, entonces ordena por la cantidad total de facts de la precondición. Esta modificación está basada en la observación de que aquellos facts que derivan de un predicado estático y no son un fact del estado inicial, entonces fallan siempre, es decir, tienen una probabilidad de fallar del 100 %. Por lo tanto, conviene colocarlos al principio del splitting por el mismo razonamiento descripto al principio de esta sección. Para terminar de comprender la importancia de esta pequeña modificación supongamos que p es un fact derivado del predicado estático $p[X]$ y p no ocurre en el estado inicial de la tarea. Entonces en la sección 3.3 vimos que las acciones que tiene a p como precondición son eliminadas de la representación proposicional de la tarea, pues son acciones no aplicables independientemente del estado. Entonces, supongamos que a es una acción que tiene a p en su precondición y deriva del esquema $a[X]$. Cuando aplicamos la operación de splitting sobre el esquema $a[X]$, ocurre que $p[X]$ está en algún sub-esquema $a_i[X_i]$. Entonces si a_i es la sub-acción que deriva de $a_i[X_i]$, tal que, p esta en la precondición de a_i, entonces a_i es eliminada de la representación proposicional de la tarea. Luego, si $a_i[X_i]$ es el último sub-esquema del splitting, esto genera la posibilidad de que el planificador aplique las $n-1$

sub-acciones anteriores del splitting, para finalmente no poder terminar de aplicar la totalidad del mismo. En cambio, si $a_i[X_i]$ es el primer sub-esquema, entonces el splitting no puede ser nunca aplicado parcialmente, ya que, su primer sub-acción fue eliminada, y de esta forma, las sub-acciones posteriores no son aplicadas en vano por el planificador.

4.5. Splitting sobre Esquemas de Acción ADL

La operación de splitting introducida en el capítulo 4 fue definida para esquemas de acción STRIPS, donde las fórmulas y efectos son conjunciones de literales y los símbolos atómicos son predicados de la forma $p[X]$. Resulta interesante extender la técnica a esquemas de acción ADL, donde las fórmulas y efectos son más generales [Pednault, 1989]. En los esquemas de acción ADL, una precondición es cualquier fórmula de la lógica de predicados. Fórmulas con los operadores booleanos habituales $(\wedge, \vee, \neg, \Rightarrow)$ y las cuantificaciones universal y existencial $(\forall, \exists)$. Por su parte, el efecto de un esquema, es una conjunción de efectos condicionales de la forma $\varphi_i \Rightarrow l_{i_1} \wedge \cdots \wedge l_{i_n}$, con φ_i una fórmula booleana definida al igual que una precondición y l_{i_j} literales. Además, cada efecto condicional puede esta alcanzado por una cuantificación universal, $\forall x_1 \ldots x_k : \varphi_i \Rightarrow l_{i_1} \wedge \cdots \wedge l_{i_n}$ [McDermott *et al.*, 1998]. La idea detrás de la extensión que vamos a definir para poder abordar estos esquemas de acción consiste en tratar estas fórmulas más generales como bloques atómicos. Para luego, poder aplicar en forma directa la mayoría de los conceptos ya estudiados en el caso de esquemas STRIPS.

4.5.1. Abstracción Black-Box

La operación de splitting sobre esquemas de acción STRIPS, aprovecha el hecho de que la precondición y el efecto de un esquema son conjunción de literales. Lo cual, resulta ser más evidente o natural el cómo dividirla en sub-esquemas. Sin embargo, no es evidente el cómo dividir fórmulas más generales en sub-esquemas, tales como, una disyunción o una cuantificación. Es por eso, que para dividir esquemas ADL trataremos a aquellos fórmulas o sub-fórmulas que no son una conjunción como un "bloque atómico indivisible". De esta manera, logramos representar las precondiciones y efectos ADL como una conjunción de estos bloques atómicos y sobre esta representación vamos a aplicar la técnica de splitting en forma similar a como lo hicimos para el caso de esquemas STRIPS. Estos bloques atómicos llamados *black-boxes*, se convierten entonces en los nuevos símbolos atómicos de las fórmulas que ocurren en las precondiciones y en los efectos de los esquemas de acción.

Supongamos que tenemos la fórmula $\varphi = (p(x) \vee q(y)) \wedge (p(x) \vee r(z))$, entonces vamos a considerar a φ como la fórmula $\varphi' = \Box_1(x, y) \wedge \Box_2(x, z)$ donde $\Box_1(x, y)$ es el black-box que abstrae la sub-fórmula $p(x) \vee q(y)$ y $\Box_2(x, z)$ el black-box que abstrae la sub-fórmula $p(x) \vee r(z)$. Aplicar la operación de splitting sobre la conjunción de bloques atómicos φ' resulta más directo y natural que sobre φ. En definitiva, vamos a reconsiderar las precondiciones y efectos ADL como una conjunción de black-boxes donde cado uno de ellos abstrae una parte de la fórmula original. Una conjunción de black-boxes $\Box_1 \wedge \cdots \wedge \Box_k$ puede ser representado mediante el conjunto $\{\Box_1, \ldots, \Box_k\}$. Por lo tanto, un esquema ADL puede ser representado mediante el conjunto de black-boxes que surgen de su precondición y de su efecto. De esta manera, mientras que en el caso de esquemas STRIPS la operación de splitting particiona los átomos anotados de un esquema. En ADL, particiona los black-boxes pertenecientes al esquema. Por supuesto, la operación de splitting sobre esquemas ADL también debe asegurar las tres propiedades descriptas en la sección 4.1. Para ello, debemos realizar algunas modificaciones al framework propuesto en la sección 4.2. Para comenzar definimos nuevamente como anotar los predicados atómicos que ocurren en un esquema de acción, en este caso cuando se trata de un esquema ADL.

Definición 37 (Atomos Anotados) *Sea $a[X]$ un esquema de acción ADL y $p[Y]$ un átomo que ocurre en $a[X]$. Entonces el* átomo anotado *correspondiente a $p[Y]$, denotado por $Ann(p[Y])$, es el átomo $p_{pre}[Y]$ si $p[Y]$ ocurre en la precondición o en una guarda de algún efecto condicional de $a[X]$; $p_{del}[Y]$ si $p[Y]$ ocurre negativamente en un efecto de $a[X]$; $p_{add}[Y]$ si $p[Y]$ ocurre positivamente en un efecto de $a[X]$.*

Dada esta nueva forma de anotar átomos de un esquema de acción, entonces el concepto de black-box se reduce al conjunto de aquellos átomos anotados que ocurren en la fórmula que es abstraída por el black-box.

Definición 38 (Black-Box) *Sea φ una precondición o efecto ADL, entonces el black-box correspondiente a φ, es el conjunto de átomos anotados $\Box_\varphi = \{Ann(p[Y]) \mid p[Y]$ ocurre en $\varphi\}$.*

Cada black-box mantiene una referencia a la precondición o efecto ADL que abstrae, ya que, esto es necesario en la etapa final de la operación de splitting donde reconstruimos las precondiciones y efectos ADL de los sub-esquemas que componen el splitting. Una vez definido el concepto de black-box, el camino a seguir es muy simple: abstraer cada precondición y efecto ADL que no es una conjunción a su correspondiente black-blox; en caso contrario, propagar la abstracción a las sub-fórmulas. De esta manera, al finalizar la propagación se obtiene un conjunto de black-boxes.

Definición 39 (Abstracción Black-Box) *Sea φ una precondición o efecto ADL, entonces la abstracción Black-Box de φ, denotada por $\mathcal{A}(\varphi)$, está definida por:*

$$\mathcal{A}(\varphi) = \begin{cases} \mathcal{A}(\varphi_1) \wedge \cdots \wedge \mathcal{A}(\varphi_n) & si\ \varphi = \varphi_1 \wedge \cdots \wedge \varphi_n \\ \square_\varphi & caso\ contrario \end{cases}$$

De la definición de $\mathcal{A}$, resulta obvio que es conveniente expresar las fórmulas en forma normal conjuntiva (CNF), pues esto permite una mayor propagación de la abstracción $\mathcal{A}$ hacia las sub-fórmulas generando una mayor cantidad de black-box's disponibles para luego ser particionados. Por otra parte, ya mencionamos que una conjunción de black-boxes puede considerado un conjunto de black-boxes. En consecuencia, un esquema ADL también es considerado un conjunto de black-boxes, denotado por $\mathcal{A}(a[X])$, producto de la unión del conjunto de black-boxes que surgen de su precondición y el conjunto de black-boxes que surgen de su efecto luego de aplicada la abstracción $\mathcal{A}$. Más precisamente, $\mathcal{A}(a[X]) = \mathcal{A}(pre(a[X])) \cup \mathcal{A}(\mathit{eff}(a[X]))$. Al ser considerado un esquema ADL un conjunto vía abstracción $\mathcal{A}$, hace que la noción de partición de esquemas ADL sea la natural. De esta manera, logramos reutilizar muchas de las ideas subyacentes del framework para hallar splitting válidos sobre esquemas de acción STRIPS. Pero con la diferencia, que en lugar de particionar un conjunto de átomos anotados, ahora particionamos un conjunto de black-boxes. Para ejemplificar todo esto, consideremos el siguiente esquema de acción ADL:

$$\begin{array}{l} A(x, y, z) \\ pre : (p(x) \vee q(y)) \wedge (p(x) \vee r(z)) \\ \mathit{eff} : (q(y) \Rightarrow \neg p(x)) \wedge \forall x : r(x) \end{array}$$

Entonces, el esquema de acción $A(x, y, z)$ vía abstracción $\mathcal{A}$ resulta en el siguiente conjunto de black-boxes:

$$A(x, y, z) = \{\square_{p(x)\vee q(y)}, \square_{p(x)\vee r(z)}, \square_{q(y)\Rightarrow\neg p(x)}, \square_{\forall x:r(x)}\}\ con$$

$\square_{p(x)\vee q(y)} = \{p_{pre}(x), q_{pre}(y)\}$,
$\square_{p(x)\vee r(z)} = \{p_{pre}(x), r_{pre}(z)\}$,
$\square_{q(y)\Rightarrow\neg p(x)} = \{q_{pre}(y), p_{del}(x)\}$,
$\square_{\forall x:r(x)} = \{r_{add}(x)\}$.

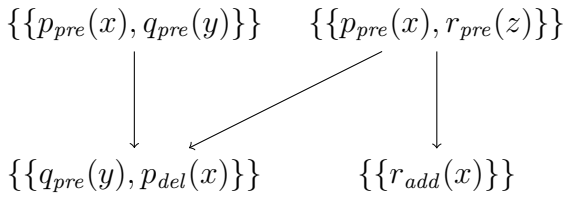
En este ejemplo, la abstracción $\mathcal{A}$ genera cuatro black-boxes y cada uno contiene los átomos anotados de la fórmula que abstrae como lo indica la definición 37.

4.5.2. Black-Box Splitting

Luego de aplicar la abstracción $\mathcal{A}$ a un esquema de acción ADL, resta particionar el conjunto de black-boxes que lo conforman y cada partición se va corresponder con un sub-esquema del splitting. Al ser cada black-box un conjunto de átomos anotados, la relación $\leq$ entre black-boxes está bien definida (ver definición 30). Extendemos la relación a conjuntos de black-boxes en la forma natural. Sean B, B' conjuntos de black-boxes, entonces $B \leq B'$ sii existen $\square_\varphi \in B$ y $\square_\psi \in B'$, tal que, $\square_\varphi \leq \square_\psi$. Por lo tanto, tenemos bien definido la relación $\leq$ entre sub-esquemas ADL. De esta manera, la definiciones de splitting (def. 29), secuencialización correcta (def. 31), splitting válido (def. 32), grafo cociente (def. 33), sub-esquemas mergeables (def. 35) y las proposiciones 9, 11 valen también para esquemas de acción ADL. Sin embargo, debemos realizar una pequeña modificación en la definición de splitting atómico y trivial de un esquema ADL.

Definición 40 *Sea $a[X]$ un esquema de acción ADL. Entonces el splitting trivial de $a[X]$, denotado por Trivial$(a[X])$, es el conjunto $\{\mathcal{A}(a[X])\}$ que asignada a cada black-box de $\mathcal{A}(a[X])$ al mismo y único sub-esquema. Y el splitting atómico de $a[X]$, denotado por Atomic$(a[X])$, es el conjunto $\{\{\square_\varphi\} \mid \square_\varphi \in \mathcal{A}(a[X])\}$ que asigna a cada black-box de $a[X]$ un sub-esquema distinto.*

A continuación, se muestra el grafo cociente del splitting atómico correspondiente al esquema $A(x, y, z)$. Cada vértice (sub-esquema) contiene un único black-box que a su vez puede contener más de un átomo anotado:



A diferencia del caso STRIPS (ver proposición 10), el splitting atómico de un esquema ADL no es necesariamente válido. Observar que dado un esquema de acción ADL $a[X]$, pueden existir black-boxes $\square_\varphi$ y $\square_\psi$ en $\mathcal{A}(a[X])$ tales que $\square_\varphi \to \square_\psi$ y $\square_\psi \to \square_\varphi$. Esto genera un ciclo en el grafo cociente del splitting atómico de $a[X]$. Sin embargo, esto es sencillo de solucionar: detectamos los ciclos que existen en el grafo cociente del splitting atómico y colapsamos los nodos (sub-esquemas) del ciclo en un único nodo (sub-esquema) [Tarjan, 1972]. El grafo cociente que se obtiene de este proceso no tiene ciclos, y por lo tanto, su splitting correspondiente es válido.

Proposición 12 *Sea $a[X]$ un esquema de acción ADL, entonces Trivial$(a[X])$ y el splitting que se obtiene a partir de Atomic$(a[X])$ eliminando todos sus cíclos, denotado con AcyclicAtomic$(a[X])$, son splitting válidos.*

Finalmente, hemos realizado las modificaciones necesarias para caracterizar los splittings válidos de un esquema de acción ADL:

Teorema 8 *Sea $a[X]$ un esquema de acción ADL, entonces cualquier splitting que se obtiene, a partir de AcyclicAtomic$(a[X])$, iterativamente fusionando sub-esquemas mergeables es válido. Viceversa, cualquier splitting válido de $a[X]$ puede ser construído de esta manera.*

Por ejemplo, si a partir del splitting atómico de $A(x, y, z)$ que es válido (no hay ciclos en su grafo cociente) y hacemos merge de los sub-esquemas $\{\{p_{pre}(x), q_{pre}(y)\}\}$, $\{\{q_{pre}(y), p_{del}(x)\}\}$ y $\{\{r_{add}(x)\}\}$, entonces obtenemos el splitting válido que particiona el esquema original en dos sub-esquemas y cuyo grafo cociente es el siguiente:

$$\{\{p_{pre}(x), r_{pre}(z)\}\}$$
$$\downarrow$$
$$\{\{p_{pre}(x), q_{pre}(y)\}, \{q_{pre}(y), p_{del}(x)\}, \{r_{add}(x)\}\}$$

Claramente, el splitting anterior solo tiene una secuencialización correcta posible. Ya habíamos mencionado que cada black-box mantiene una referencia a la precondición o efecto que abstrae, la cual, es utilizada para recuperar las precondiciones y efectos de cada sub-esquema que componen el splitting:

$A_1(x, z)$	$A_2(x, y)$
$pre : p(x) \vee r(z)$	$pre : p(x) \vee q(y)$
$eff :$	$eff : (q(y) \Rightarrow \neg p(x)) \wedge \forall x : r(x)$

Observar que el framework introducido en la sección 4.2 para hallar splitting válidos sobre esquemas de acción STRIPS puede ser visto como un caso particular del framework para hallar splitting válidos sobre esquemas de acción ADL. Esto si consideramos cada átomo anotado $Ann(p[X]) \in a[X]$ como un black-box que contiene un único elemento, i. e., un singleton $\square_{p[X]} = \{Ann(p[X])\}$. Por lo tanto, podemos decir que en esta sección describimos un framework para hallar splitting válidos aún más general. Por último, mencionamos que la decoración de un splitting válido de un esquema ADL es exactamente el mismo que ya definimos en la sección 4.3. Mientras que la aproximación por Hill Climbing de la sección 4.4.2, se mantiene prácticamente sin cambios, salvo que, en lugar de iniciar la búsqueda partiendo desde $Atomic(a[X])$, lo hacemos partiendo desde $AcyclicAtomic(a[X])$.

4.6. La noción de Spliteabilidad

En esta sección presentamos la noción de *spliteabilidad* asociada a un esquema de acción, el cual, surge como un intento de determinar que tipos de esquemas de acción resultan ser más optimizables por la operación de splitting. La idea detrás de esta noción es capturar ciertas características estructurales de un esquema que nos permita predecir la probabilidad de que el esquema sufra una importante optimización al ser sometido a la operación de splitting. En definitiva, nuestra intención es dar una medida de *cuán spliteable* es un esquema de acción. Este concepto está basado en aspectos estructurales en los esquemas tales como la cantidad de variables en su interfaz y la forma en que estas variables están relacionadas por las interfaces de los diferentes átomos que componen el esquema. Estos aspectos permiten capturar en forma general qué esquemas del dominio requieren ser optimizados con mayor necesidad por nuestra técnica de splitting. Por ejemplo, supongamos que tenemos un esquema de acción $a_1[X_1]$ con una única variable en su interfaz ($|X_1| = 1$) y otro esquema $a_2[X_2]$ con tres variables en su interfaz ($|X_2| = 3$). Es evidente que el esquema $a_2[X_2]$ es el único de los dos esquemas que su interfaz puede ser optimizada por nuestra técnica. En general, dado un conjunto de esquemas $a_1[X_1] \dots a_n[X_n]$, el concepto de spliteabilidad define un orden total sobre los esquemas según que tan optimizables vía splitting pueden llegar a ser. En otras palabras, la idea de spliteabilidad puede ser entendida como una herramienta para estimar qué esquemas de un dominio pueden generar a priori una mayor cantidad de acciones proposicionales, ya que esto esquemas, suelen ser aquellos que provocan que el proceso de grounding no finalice completamente como consecuencia de agotar los recursos computacionales (tiempo y memoria) disponibles para la ejecución del planificador. Al mismo tiempo que captura aquellos esquemas que poseen una estructura que hace posible que la operación de splitting sobre ellos produzca efectivamente una considerable mejora en la performance del proceso de grounding. El concepto de spliteabilidad propuesto en esta sección extrae información contenida en el dominio de la tarea sin utilizar información proveniente del problema en particular. Esto se debe a que decimos que la optimización vía splitting de un dominio sea general para cualquier problema arbitrario. Por último, vamos a extender de una forma muy simple la noción de spliteabilidad a nivel de dominios, y de esta manera, lograr contar con una métrica para comparar el nivel de spliteabilidad entre dominios distintos, caracterizando aquellos que tienen mayor probabilidad de ser exitosamente optimizados por nuestra técnica.

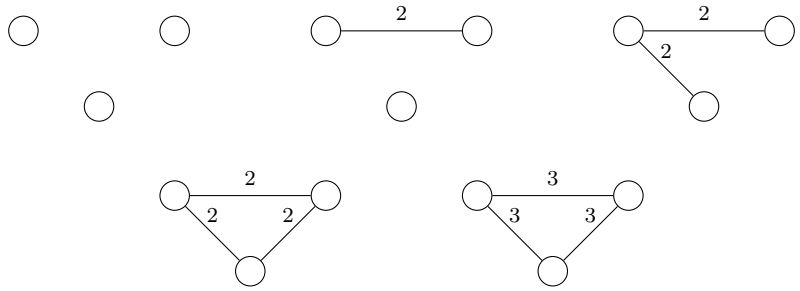
4.6.1. El grafo de spliteabilidad

Para caracterizar aquellos esquemas más compatibles para ser optimizados por la operación de splitting, proponemos definir una representación más abstracta de los mismos. Para ello, representamos un esquema de acción como un grafo que muestra como las variables están relacionadas por los átomos que componen el esquema. Básicamente, este grafo permite abstraer de un esquema de acción las características consideradas más importantes para la obtención de un splitting interesante del esquema. Los nodos del grafo son las variables que componen la interfaz del esquema y existe una arista entre dos variables si ambas ocurren en la interfaz de un átomo. Además, una arista está etiquetada con la aridad del átomo con mayor aridad que contiene a ambas variables en su interfaz.

Definición 41 (Grafo de Spliteabilidad) *Sea $a[X]$ un esquema de acción. Entonces su grafo de spliteabilidad es el grafo etiquetado $\mathcal{G}(a[X]) = (V, E)$ con $V = X$ y $E = \{(x, y, k)) \mid x \neq y \in X'$ para algun $p[X'] \in At(a[X])$ y $k = \text{máx}_{p[X']\in At(a[X])}\{|X'| : x \neq y \in X'\}\}$.*

Para todo grafo de splitabilidad $\mathcal{G} = (V, E)$ con n nodos existen dos casos extremos con respecto a su nivel de spliteabilidad: 1) cuando $E = \emptyset$, llamado grafo vacío, y cuando $E = \{(x, y, n) \mid x \neq y \in V\}$, llamado grafo completo. El grafo vacío indica que es posible particionar el esquema original hasta inclusive tener un sub-esquema por cada variable, es decir, es posible reducir el tamaño de la interfaz original del esquema (n) a 1. Mientras que el grafo completo indica que el esquema no permite obtener ninguna partición del mismo que reduzca el tamaño de la interfaz original. De esta manera, el grafo vacío tiene el mayor nivel de spliteabilidad y el grafo completo el menor. Cualquier otro grafo de n nodos se ubica entre estos dos casos extremos con respecto al nivel de spliteabilidad, ya que, cualquier otro grafo (con n nodos) debe tener más aristas que el grafo vacío y menos que el completo. Definimos $\mathcal{G}^n = \{g \mid g$ *es un grafo de spliteabilidad con n nodos*$\}$ y $\mathcal{G}^\infty = \bigcup_{n=1}^{\infty} \mathcal{G}^n$. Notar que los nombres de las variables no son relevantes, por lo tanto, podemos consideramos $\mathcal{G}^n$ y $\mathcal{G}^\infty$ módulo isomorfismo. Entonces queremos definir un orden total sobre $\mathcal{G}^\infty$. Para ello, y para ganar intuición, primero proponemos observar los grafos que pertenecen a $\mathcal{G}^n$ con valores pequeños de n. Por ejemplo, cualquier esquema de acción con dos variables en su interfaz ($n = 2$) tiene su grafo de spliteabilidad en $\mathcal{G}^2$ y en $\mathcal{G}^2$ solamente existen dos grafos, el vacío y el completo:

El grafo vacío de $\mathcal{G}^2$ puede ser dividido únicamente en dos sub-esquemas con interfaz x e y, respectivamente. Mientras que el grafo completo de $\mathcal{G}^2$ carece de un splitting interesante, pues cualquier splitting posible tiene al menos un sub-esquema con la misma interfaz que el esquema original, en consecuencia, la operación de splitting no tiene ninguna posibilidad de reducir el tamaño de la interfaz del esquema original. A su vez, para el caso de un esquema de acción con tres variables en su interfaz, su grafo de spliteabilidad debe ser alguno de los grafos en $\mathcal{G}^3$:



En $\mathcal{G}^3$ existen otros grafos distintos del grafo vacío y completo. Se observa que los grafos que tienden a acercarse más al grafo completo, con respecto a la cantidad de aristas y sus etiquetas, son aquellos donde la técnica de splitting difícilmente encuentre una partición del esquema original que logre reducir el tamaño de la interfaz original. Entonces, generalizando podemos concluir que para grafos con n nodos existe una orden de spliteabilidad comenzando por el grafo vacío y terminando en el grafo completo de $\mathcal{G}^n$.

Hasta este momento, hemos comparado grafos con la misma cantidad de nodos (variables). Pero también queremos comparar grafos con diferente cantidad de variables. Esta tarea no es sencilla, por ejemplo, supongamos que tenemos dos esquemas de acción con los siguientes grafos de spliteabilidad en $\mathcal{G}^4$ y $\mathcal{G}^5$, respectivamente:

Para este ejemplo, no es evidente o fácil de concluir cuál de los dos grafos es más spliteable. Por lo tanto, proponemos un método de decisión que consiste en extraer ciertas aspectos relevantes del grafo de spliteabilidad para ser ponderados mediante una fórmula y esta finalmente devuelve el nivel

de spliteabilidad del grafo. Para esto, resulta práctico definir el conjunto de todos los grafos vacíos $\mathcal{G}_\emptyset = \{(V, E) \in \mathcal{G}^\infty \mid E = \emptyset\}$ y el conjunto de todos los grafos completos $\mathcal{G}_C = \{(V, E) \in \mathcal{G}^\infty \mid E = V \times V \times \{|V|\}\}$.

4.6.2. La función de spliteabilidad

Una función de spliteabilidad es una función $\mathcal{F} : \mathcal{G} \to [0, 1]$ que pondera determinados atributos de un grafo de spliteabilidad $g = (V, E)$ correspondiente a un esquema de acción y devuelve un valor que representa el nivel de spliteabilidad del mismo. De esta manera una función de spliteabilidad $\mathcal{F}$ define un orden total sobre $\mathcal{G}$ en la siguiente forma: sean $a[X]$ y $a'[X']$ esquemas de acción, diremos que $a'[X']$ es más spliteable que $a[X]$, denotado por $a[X] \leq_\mathcal{F} a'[X']$ sii $\mathcal{F}(\mathcal{G}(a[X])) \leq \mathcal{F}(\mathcal{G}(a'[X']))$. Consideramos que una definición razonable de una función de spliteabilidad $\mathcal{F}$ debe satisfacer al menos las siguientes propiedades:

- Un grafo completo nunca puede ser optimizado por la operación de splitting, por lo tanto, $\mathcal{F}(g) = 0$ para todo $g \in \mathcal{G}_C$.

- Entre los grafos vacíos, aquellos con mayor cantidad de nodos deben ser más spliteables que aquellos con menor cantidad, por lo tanto, $\lim_{|V|\to\infty} \mathcal{F}(g) = 1$ con $g \in \mathcal{G}_\emptyset$.

- $\mathcal{F}$ debe asignar mayor spliteabilidad a aquellos esquemas con interfaces más grandes debido a que tienden a generar una mayor cantidad de acciones proposicionales. Entonces maximizar el término $1 - \frac{1}{|V|}$ implica mayor spliteabilidad.

- $\mathcal{F}$ debe asignar mayor spliteabilidad a los grafos con mayor cantidad de componentes conexas, denotado por $\mathcal{CC}(g)$, debido a que estos pueden tener splittings eficientes en el sentido de que las interfaces de los sub-esquemas no presentan variables en común. Entonces maximizar el término $1 - \frac{1}{\mathcal{CC}(g)}$, implica mayor spliteabilidad.

- $\mathcal{F}$ debe asignar mayor spliteabilidad a los grafos con menor cantidad de aristas, es decir, con baja densidad. Esto surge de la observación de que los átomos de un esquema hacen inseparables a las variables que ocurren su interfaz. Una mayor presencia de aristas en el grafo implica una menor posibilidad de separación de las variables del esquema. En particular, definimos la densidad de un grafo de spliteabilidad mediante

la fórmula:

$$\mathcal{D}(g) = \frac{\sum\limits_{(x,y,k)\in E} k}{\sum\limits_{x\neq y\in V} |V|}$$

Entonces maximizar el término $1-\mathcal{D}(\mathcal{G})$, implica mayor spliteabilidad.

Dadas las propiedades que consideramos debe cumplir una función de spliteabilidad, entonces proponemos la siguiente:

Definición 42 (Función de Spliteabilidad) *Sea $\mathcal{F} : \mathcal{G} \to [0,1]$ la función de spliteabilidad definida por la siguiente ecuación:*

$$\mathcal{F}(g) = \begin{cases} 1 - \frac{1}{3} \cdot \left[\frac{1}{|V|} + \frac{1}{\mathcal{CC}(g)} + \mathcal{D}(g)\right] & \text{si } g \notin \mathcal{G}_C \\ 0 & \text{si } g \in \mathcal{G}_C \end{cases} \tag{4.2}$$

La función de spliteabilidad propuesta es una ponderación lineal de tres atributos del grafo: cantidad de vértices, cantidad de componentes conexas y su densidad. Cuando el valor devuelto por $\mathcal{F}$ tiende a 1 eso indica que el esquema de acción presenta una estructura propicia para que mediante la operación de splitting se logre reducir la cantidad acciones instanciadas que origina el esquema. En el otro extremo, cuando el valor devuelto por $\mathcal{F}$ tiende 0 indica que el esquema no cuenta con una estructura adecuada para ser optimizada por la operación de splitting, y por lo tanto, resultaría poco probable obtener una disminución en la cantidad de acciones instanciadas del esquema.

Extendemos la noción de spliteabilidad a dominios, utilizando la noción ya definida a nivel de esquemas de acción, de forma tal de poder caracterizar aquellos dominios para los cuales la técnica de splitting tiene más posibilidades de causar un efecto positivo. Definimos la spliteabilidad de un dominio como la máxima spliteabilidad entre sus esquemas de acción: sea $D = \{a_1[X_1], \ldots, a_n[X_n]\}$ un conjunto de esquemas de acción i. e., un dominio, entonces la función de spliteabilidad de D está definida por $\mathcal{F}(D) = \text{máx}_i\, \mathcal{F}(a_i[X_i])$. Esta definición está basada en la observación de que es suficiente con la existencia de al menos un esquema de acción en el dominio con un alto nivel de spliteabilidad para que la operación de splitting sobre este esquema lleve a obtener un mejor resultado en comparación al dominio original. Una alternativa más natural hubiese sido tomar el promedio de las spliteabilidades de los esquemas, i. e., $\mathcal{F}(D) = \frac{\mathcal{F}(a_i[X_i])}{n}$. Pero el promedio hace que los esquemas de acción con baja spliteabilidad produzacan una disminución importante en la spliteabilidad del dominio. Esto puede

resultar en un efecto no deseado, ya que, muchos dominios pueden presentar un único esquema de acción con alta spliteabilidad que es el que produce finalmente una gran cantidad de acciones proposicionales en el proceso de grounding, mientras que el resto de los esquemas no representan un verdadero problema de grounding. Esta situación es salvada si consideramos la máxima spliteabilidad de los esquemas y no el promedio.

En la sección 5.6 del capítulo 5 validamos la función de spliteabilidad propuesta en la ecuación 4.2, donde estudiamos que sucede si aplicamos la operación de splitting sólo en aquellos esquemas de acción que presentan el mayor valor de splitabilidad contra una elección aleatoria de esquemas con el objetivo de concluir empíricamente si nuestra noción de spliteabilidad señala los esquemas del dominio donde mediante splitting se logra una considerable reducción en la cantidad de acciones proposicionales generadas durante el proceso de grounding del planificador.

Capítulo 5

Evaluación: experimentos y resultados

En este capítulo se presentan los resultados empíricos obtenidos de los experimentos realizados con nuestra técnica *Action Schema Splting*. Estos experimentos están enfocados principalmente en estudiar si la técnica produce una disminución significativa en la cantidad de acciones que se generan en el proceso de grounding del planificador. Además, analizamos cómo la reformulación PDDL del problema, producto de la división de esquemas de acción en sub-esquemas, repercute finalmente en el proceso de búsqueda heurística. Nuestra técnica de splitting, fue desarrollada a partir del parser del planificador Fast Forward [Hoffmann and Nebel, 2001a], el cual está implementado en lenguaje C. El código fuente de nuestro algoritmo se encuentra disponible para descargar en un repositorio `github`[1], junto con los dominios utilizados en este capítulo de evaluación.

5.1. Acerca del benchmarks

El primer paso para poder efectuar los experimentos, con el objetivo de estudiar el comportamiento de nuestra técnica, consiste en decidir *sobre qué dominios realizar los experimentos*. Lo natural para el área de planning, es seleccionar un conjunto de dominios IPC, es decir, dominios que fueron parte del benchmark de alguna edición de las competencias internacionales de planning. Sin embargo, es sabido que, estos dominios son diseñados especialmente para ser un desafío de búsqueda y no del proceso de grounding. Por tanto, son cuidadosamente pensados para que el tamaño de la representación proposicional de la tarea no sea un obstáculo para el planificador, y

[1] https://github.com/facubus/ActionSchemaSplitting.git

siempre se pueda ejecutar la búsqueda sobre dicha representación. De hecho, dominios IPC tales como Pipesworld y Cybersecurity [Hoffmann *et al.*, 2006; Boddy *et al.*, 2005], son el resultado de dividir manualmente sus esquemas de acción originales. Esto se realizó para lograr una versión que no tenga problemas en el proceso de grounding del planificador.

Lamentablemente, no existe un benchmark destinado especialmente a evaluar capacidades de grounding. Los dominios IPC se caracterizan en general por tener esquemas con interfaces pequeñas, hasta 4 ó 5 variables en el peor de los casos. Por consiguiente, los esquemas son muy poco divisibles, y por ende, es razonable que no haya mucho por ganar al aplicar la técnica de splitting. Sin embargo, los dominios IPC pueden ser interesantes para estudiar como la técnica afecta la búsqueda heurística, en relación con el aumento de la distancia a la meta que produce el reemplazo de acciones por sus respectivas sub-acciones.

Con el afán de evaluar nuestra técnica, fuimos capaces de conseguir unos pocos dominios que no fueron diseñados para una competencia IPC en particular. En realidad, fueron diseñados para modelar un determinado problema de alguna disciplina específica, para ser resueltos con las técnicas de planning. Estos dominios se caracterizan por tener esquemas de acción con interfaces más grandes que los esquemas habituales que podemos encontrar en las distintas ediciones de IPC. Por lo cual, este tipo de dominios en un benchmark mucho más interesante para evaluar nuestra técnica de optimización.

En lo que resta del capítulo, cuando hablemos de "dominios PPC" (Pre-Process Challenge) nos referiremos a este tipo de dominios y con "dominios IPC" (International Planning Competition) a los dominios tradicionales obtenidos de las diferentes competencias internacionales de planning. Aclaramos que, los dominios PPC no son fáciles de hallar, por lo cual, no disponemos de una gran cantidad, mientras que si contamos con una vasta cantidad de dominios IPC. En este sentido, 27 fueron los dominios que formaron parte de nuestro benchmark IPC, mientras que solo dos PPC. Al dominio Pipesworld, lo consideramos PPC, ya que, contamos con su versión original y su versión "dividida a mano"(versión IPC). Esto lo convierte en un dominio atractivo porque nos va a permitir comparar si los resultados de nuestro splitting automático son competitivos con respecto a los arrojados por el splitting manual. Esto resulta muy interesante si consideramos la versión con splitting manual, como una optimización realizada por un diseñador experto en el dominio. Otro dominio PPC, será el domino STRIPS "Genome Edit Distance"(GED) de Haslum [Haslum, 2011], el cual, también cuenta con una versión original (ged3-itt.pddl) y una versión manualmente spliteada (ged2-itt.pddl) con el conjunto de tareas "ds1" provisto por el autor. En definitiva, lanzaremos experimentos sobre el conjunto de dominios IPC y PPC con el objetivo

	Trivial			Manual			HC 0.8			HC 0.7			HC 0.0			Atomic		
	ac	avg	mx	ac	avg	mx	ac	avg	mx	ac	avg	mx	ac	avg	mx	ac	avg	mx
GED	14	2.4	3	21	1.8	**2**	24	2.0	3	26	1.9	**2**	26	1.9	**2**	163	**1.1**	2
Pipesworld	4	8.0	9	6	6.3	7	8	5.0	7	10	4.6	5	24	2.7	**3**	59	**2.0**	3
Freecell	10	4.9	7	-	-	-	19	2.7	7	24	2.2	5	35	1.9	**2**	117	**1.3**	2
Transport	3	4.3	5	-	-	-	Trivial			Trivial			15	**2.0**	**2**	20	**2.0**	2

Cuadro 5.1: Estadísticas de los efectos del Splitting sobre la representación PDDL de los dominios listados. "ac": cantidad de esquemas de acción; "avg": promedio tamaño interfaces; "mx": máximo tamaño de interfaz. Valores mínimos resaltados en negrita.

de analizar los efectos del splitting sobre estas dos clases de dominios.

5.2. Efectos del Splitting en el Dominio

En esta sección evaluamos los efectos que produce el splitting sobre la descripción PDDL de los dominios, tales como el aumento en la cantidad esquemas o la disminución en el tamaño de las interfaces. Para esto el experimento consiste en aplicar splitting sobre los benchmarks IPC y PPC, con valores de $\gamma \in \{0, 0.1, 0.2, \ldots, 0.9\}$, para la implementación por Hill Climbing (HC) explicada en la sección 4.4.2. En el Cuadro 5.1 mostramos las estadísticas más importantes sobre los efectos que produce el splitting en los dominios IPC: *Freecell* y *Transport*; y los dominios PPC: *GED* y *Pipesworld*. Mostramos para cada dominio: la cantidad de esquemas, el promedio de las interfaces de los esquemas y el tamaño máximo entre las interfaces de los esquemas. Como podemos ver en el cuadro, los efectos del splitting sobre la representación PDDL de los dominios, varían de acuerdo al nivel de granularidad del algoritmo, que es regulado por el parámetro γ. Decidimos no reportar tiempos de ejecución por ser despreciables (a lo sumo un par de segundos). Por último, solo se reporta estadísticas para $\gamma \in \{0, 0.7, 0.8\}$, ya que, para otros valores no se obtuvieron dominios distintos.

Luego de analizar el Cuadro 5.1, podemos verificar que:

- El splitting trivial es óptimo con respecto a la cantidad de esquemas, pero, presenta los peores valores para el promedio (avg) y máximo tamaño de interfaz (mx).

- Por su parte, HC para valores mas pequeños de γ obtiene tamaños de interfaces más pequeños, pero, con el efecto no deseado de generar más esquemas.

- El splitting atómico es óptimo con respecto al promedio y máximo tamaño de interfaz, pero, al mismo tiempo es el peor caso con respecto a la cantidad de esquemas. Esto es consistente porque sabemos, por el capitulo 4, que ningún splitting válido puede mejorar la interfaz del splitting atómico.

- En el dominio *Transport* para HC con $\gamma = 0.8$ y 0.7 el algoritmo no alcanza a dividir ningún esquema, es decir, converge al splitting trivial.

- Con respecto al splitting manual, se observa que es más suave inclusive que el menos agresivo de los splittings automáticos, en el sentido de que producen 21 y 6 esquemas contra 24 y 8 de HC con $\gamma = 0.8$ en *GED* y *Pipesworld*, respectivamente.

- En conclusión, para estos dos dominios el splitting automático más parecido al splitting manual es el obtenido mediante HC con $\gamma = 0.8$.

- En el dominio *Freecell* debemos tomar un valor de $\gamma = 0.7$ para recién obtener una disminución en la máxima interfaz del dominio pagando el costo de tener 14 esquemas más con respecto al dominio trivial (de 10 a 24).

- Para el caso del dominio *Transport* recién con un valor de $\gamma = 0.0$ se consigue un dominio distinto al trivial, con el efecto negativo de aumentar el número de esquemas de 3 a 15 pero con el efecto positivo de reducir la máxima interfaz de 5 a 2.

- Los efectos producidos por el splitting automático dependen del valor elegido para el parámetro γ. Y a su vez, del dominio en particular, ya que, algunos con un valor más pequeño de γ no se ven tan afectados como otros.

Para completar el análisis, en el Cuadro 5.2 se pretende visualizar de una forma más concisa si un splitting optimiza mejor el dominio trivial que otro. Para ello, consideramos la proporción entre la diferencia en la cantidad de esquemas de un splitting y el dominio trivial (Δ ac) sobre la diferencia entre el tamaño máximo de sus interfaces (Δ mx). Mientras más bajo es este valor, significa que el splitting logró reducir el valor de "mx" aumentando muy poco el valor de "acc". Esto señala, que el splitting es una muy buena optimización del dominio trivial. Cuando un splitting no mejora el valor "mx"del dominio trivial, entonces para evitar la división por cero utilizamos ∞ para indicar que no hubo optimización alguna. En resumen, del Cuadro 5.2 podemos ver que:

	Manual	HC 0.8	HC 0.7	HC 0.0	Atomic
GED	7	∞	12	12	149
Pipesworld	1	2	1,5	3,3	9,1
Freecell	-	∞	7	5	21,4
Transport	-	Trivial	Trivial	4	5,6

Cuadro 5.2: Proporción $\frac{\Delta ac}{\Delta mx}$ para los dominios obtenidos por splitting automático y manual.

- La versión manual tiene los mejores valores en los dominios donde este está definido.
- En *GED*, el splitting automático con el valor más cercano a manual fue HC con $\gamma = \{0.7, 0.0\}$.
- En *Pipesworld*, HC con $\gamma = 0.7$ logra obtener una proporción muy cercana a manual (de 1 a 1.5) y también se puede destacar HC con $\gamma = 0.0$ con valor 2.
- Para el dominio *Freecell*, HC con $\gamma = 0.0$ alcanza la mejor proporción con un valor de 5.

Concluimos entonces, que los splittings automáticos son capaces de obtener un buen compromiso entre el aumento de esquemas y la reducción de sus interfaces. En general, para valores no extremos de γ. Esto a tal punto de que los splittings obtenidos por el algoritmo son competitivos, incluso respecto de una versión del dominio producto de un splitting manual, efectuado por un experto.

5.3. Efectos del Splitting en el Grounding

En esta sección estudiamos los efectos del splitting en el rendimiento del proceso de grounding del planificador Fast Downward [Helmert, 2006]. Este planificador es considerado hoy, un estándar en el área, ya que incluye la mayoría de las técnicas de búsqueda más competitivas. Los resultados que reportamos en esta Sección, corresponden al grounding por alcanzabilidad relajada, vista en la sección 3.4. La totalidad de los experimentos que mostramos, se realizan sobre los dominios IPC y PPC. Por cuestiones de simplicidad, en adelante, a la cantidad de acciones generadas por el proceso de grounding, la mencionaremos simplemente como "grounding".

Comenzaremos con la figura 5.1, donde se muestra el grounding correspondiente a los dominios IPC. Cada punto se corresponde con una tarea de

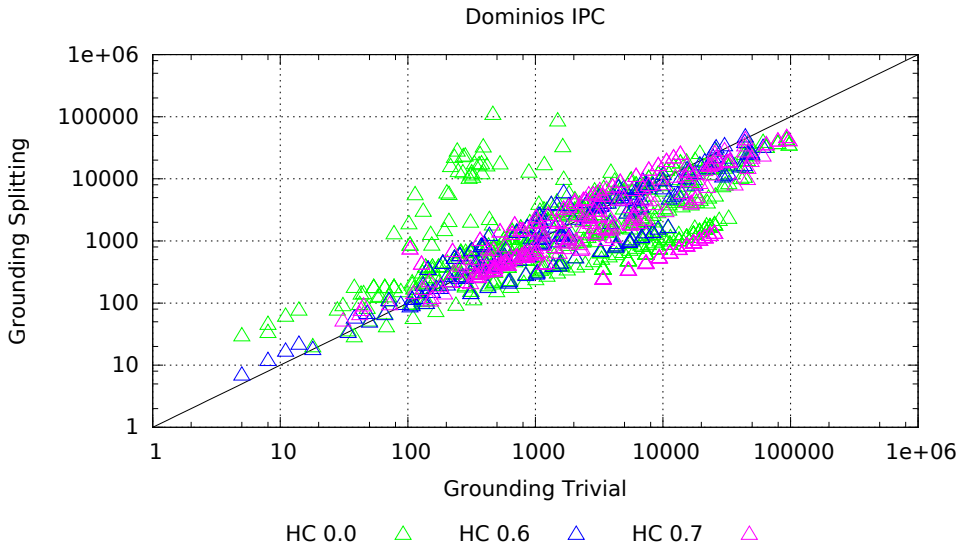


Figura 5.1: Grounding para Trivial splitting, vs splitting por HC con $\gamma \in \{0, 0.5, 0.6, 0.7, 0.8\}$ en los dominios IPC.

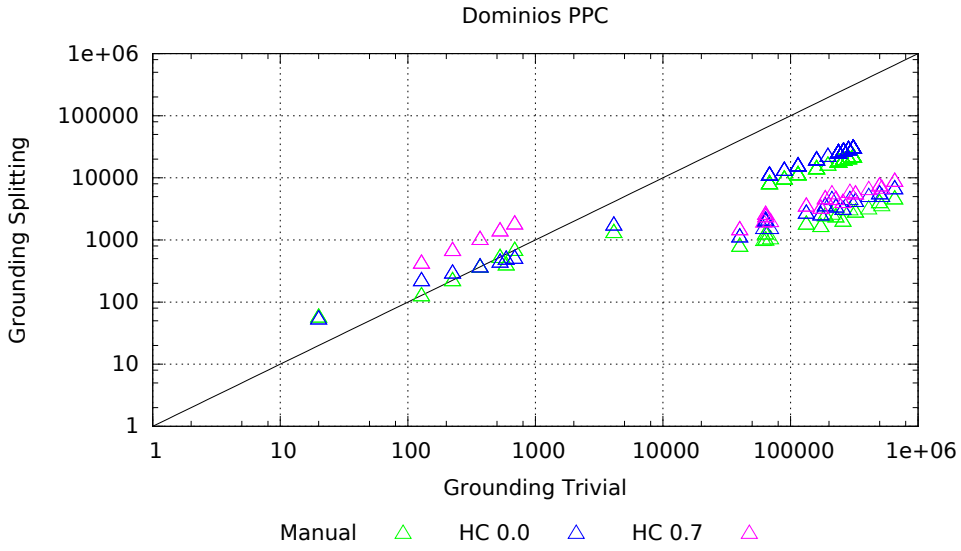


Figura 5.2: Grounding para Trivial y Manual splitting, vs splitting por HC con $\gamma \in \{0, 0.7, 0.8\}$ en dominios PPC.

un dominio en particular. En el eje horizontal representamos el grounding correspondiente al dominio trivial (sin dividir) y en el eje vertical el grounding del splitting obtenido mediante diferentes valores de γ.

La primera observación que podemos hacer es que, el grounding de las tareas evaluadas, está siempre por debajo de las 100 mil acciones para el dominio original. Esto evidencia una vez más, que los dominios IPC no se caracterizan por ser un desafío de grounding, sino de búsqueda como mencionamos en la Sección 5.1. Sin embargo, también se observa que existe una mayor concentración de puntos por debajo de la diagonal, lo que indica, que en muchas de las tareas el grounding de los dominios con splitting, disminuye con relación al dominio trivial. De hecho, mientras el grounding de los dominios originales, considerando todos los dominios y tareas del benchmark, se aproxima a los 4 millones acciones, el de los dominios procesados con el splitting mas agresivo (HC con $\gamma = 0$), gira en torno a las 2.5 millones. Por desgracia, veremos más adelante, que no se trata de una reducción lo suficientemente importante, para obtener una mejora general en el proceso de búsqueda. Esto se puede explicar, porque la optimización del grounding observada, no alcanza para pagar el costo de aumentar la distancia a la meta, producido por el reemplazo de acciones por sus respectivas sub-acciones.

Necesitamos una reducción muy importante del grounding para que se justifique el alejamiento de la meta. Es importante notar que, ya mencionamos que los dominios IPC, por su construcción, no dejan mucho lugar para ser optimizados por splitting. Para aportar un poco más de luz, la figura 5.2 muestra los valores de grounding sobre el conjunto de dominios PPC. Como vemos, la optimización, es mayor a la observada para los dominios IPC. En las tareas más grandes del benchmark, hay una mejora de hasta dos ordenes de magnitud. Tareas que contaban con 100 mil a 1 millón de acciones en el caso trivial, utilizando splitting, se reducen a tener entre mil y 10 mil acciones. Esto no sorprende, porque, ya mencionamos que los esquemas de acción de los dominios PPC son más propicios para ser optimizados por nuestra técnica, fundamentalmente *Pipesworld*.

Además, se observa que los splittings automáticos, en muchas tareas logran reducir el grounding considerablemente, incluso, por encima del splitting manual. Y en otras tareas, ambas alternativas, optimizan el grounding en forma similar. Esto significa, que el splitting automático, al menos sobre los dominios PPC, pueden representar una alternativa superadora (en grounding) al splitting manual. Sin embargo, aún resta observar, si esta optimización en el grounding es suficiente para neutralizar la contra parte del efecto negativo que produce el splitting en el espacio de búsqueda. Dicho análisis sigue a continuación.

5.4. Efectos del Splitting en la Búsqueda

En esta sección estudiamos los efectos de la técnica de splitting en el rendimiento del proceso de búsqueda del planificador. El análisis, lo realizaremos para los experimentos de la sección anterior. Para tal fin, ejecutamos dos búsquedas con heurísticas distintas. Una de ellas, es la heurística h^{FF} en modo *lazy greedy best-first search* sin *preferred operators* y la otra conocida como *primera iteración LAMMA* [Richter and Westphal, 2010], considerada estado del arte con respecto a tiempo de ejecución.

En la Figura 5.3, comparamos la cantidad de estados expandidos[2] por el proceso de búsqueda para la heurística h^{FF}, entre el dominio trivial y los diferentes splittings. La masa de puntos se ubica muy por encima de la diagonal, indicando que en los dominios con splitting, se necesita expandir mucho más estados para alcanzar la meta, en comparación, con los necesarios para el dominio trivial. Si bien un lector avezado, podría argumentar que el deterioro en el rendimiento de la búsqueda no necesariamente se debe al efecto del splitting sobre el dominio, sino, a la heurística utilizada puntualmente (h^{FF}). Tal vez, esta heurística no es apropiada para realizar búsquedas en espacios de estados con splitting. Es por esto, que en la figura 5.4 mostramos los estados expandidos para la heurística LAMA. Podemos ver que se presenta el mismo fenómeno, es decir, una gran concentración por encima de la diagonal. Esto fortalece la hipótesis de que el deterioro de la búsqueda, no es producto de la heurística que consideremos, sino, originada por los efectos propios de la técnica de splitting, en el espacio de búsqueda de la tarea.

Es importante notar que, en las gráficas exhibidas, la cantidad de estados expandidos aumentan a media que más agresivo es el splitting. Por ejemplo, para $\gamma = 0$ hay una gran concentración de puntos en la parte superior izquierda. Mientras que, para splittings más suaves, como por ejemplo para *gamma* $\in \{0.7, 0.8\}$ los puntos tienden ubicarse más cerca de la diagonal. Esto sugiere fuertemente que, debemos dividir los esquemas cuidadosamente, ya que, un abuso de sub-acciones, si bien, puede mejorar el proceso de grounding, tiene como contra parte, el efecto de complicar el proceso de búsqueda del planificador.

Para entender cuando compensa pagar el costo que splitting tiene en la búsqueda, en la figura 5.5 se muestran los estados expandidos por la búsqueda con heurística h^{FF}en los dominios PPC. A simple vista, los splitting continúan deteriorando el proceso de búsqueda, pero, en comparación con la misma gráfica para los dominios IPC, el deterioro parece ser menor. En defi-

[2]Los cantidad estados expandidos, es estándar en el área, para evaluar la performance de la búsqueda de los planificadores.

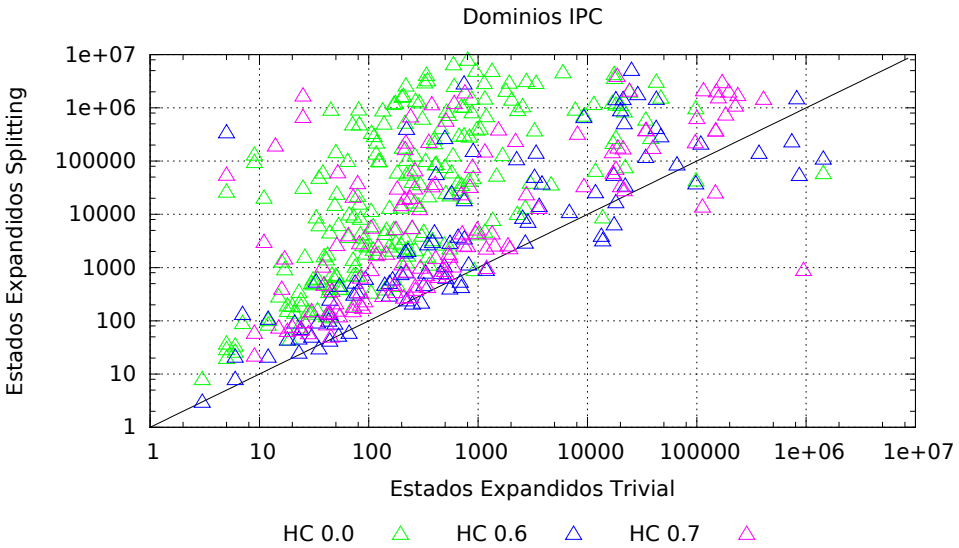


Figura 5.3: Estados expandidos por la heurística de búsqueda h^{FF} para Trivial vs Splitting por HC con $\gamma \in \{0, 0.5, 0.6, 0.7, 0.8\}$ en los dominios IPC.

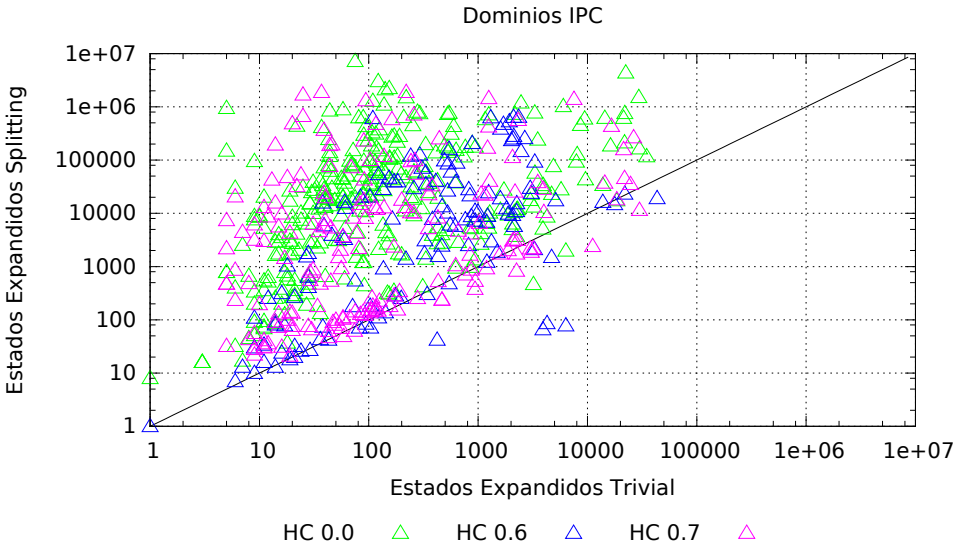


Figura 5.4: Estados expandidos por la heurística de búsqueda LAMA para Trivial vs Splitting por HC con $\gamma \in \{0, 0.5, 0.6, 0.7, 0.8\}$ en los dominios IPC.

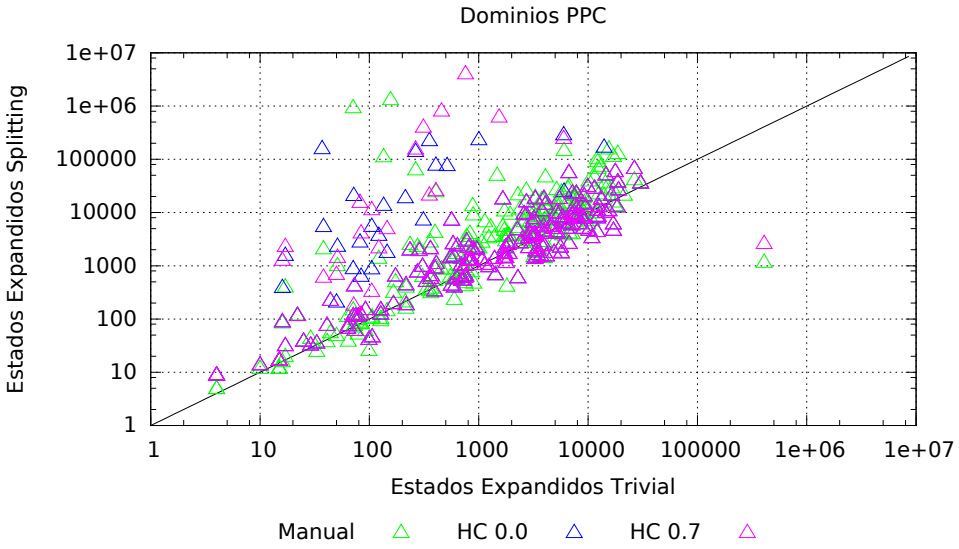


Figura 5.5: estados expandidos para Trivial y Manual splitting, vs splitting por HC con $\gamma \in \{0, 0.7, 0.8\}$ en dominios PPC.

nitiva, splitting reduce el grounding, pero, siempre con un costo en búsqueda. Por lo tanto, podemos decir que la técnica debe ser aplicada en la medida justa y necesaria si deseamos, además de reducir el grounding, tener ciertas posibilidades de hallar un plan en la búsqueda.

Finalmente, con el objetivo de evaluar la compensación del uso de splitting, en el cuadro 5.3 mostramos la cantidad de tareas resueltas utilizando el dominio original, manual y los diferentes splittings. Este valor se denomina *coverage*, y dentro del área de planning se utiliza como un estándar para evaluar las performances de nuevas técnicas[3]. De los cuadros anteriores podemos notar que:

- Para el caso de la heurística h^{FF}, en el dominio *GED* logramos mejorar el coverage con respecto al dominio trivial (de 15 a 66) e inclusive con respecto al manual (de 52 a 66), a través de HC con $\gamma = \{0.7, 0.8\}$.

- En el dominio *Pipesworld* el splitting con $\gamma = 0.8$ logra resolver 3 tareas más (de 25 a 28) que el trivial e iguala el coverage del manual, es decir, nuestro splitting automático optimizó el dominio con la misma efectividad conseguida por un experto del dominio.

[3]Es importante recordar, que los dominios de IPC *Freecell* y *Transport* no cuentan con una versión manual

h^{FF}							
Dominio	#	Trivial	Manual	HC 0.8	HC 0.7	HC 0.0	Atomic
GED	110	15	52	21	**66**	**66**	6
Pipesworld	50	25	**28**	**28**	17	10	13
Freecell	60	**59**	-	19	24	30	39
Transport	30	**30**	-	Trivial	**30**	25	26
LAMA							
Dominio	#	Trivial	Manual	HC 0.8	HC 0.7	HC 0.0	Atomic
GED	110	73	**110**	34	76	76	6
Pipesworld	50	38	**44**	**44**	23	23	17
Freecell	60	**59**	-	36	38	35	31
Transport	30	**30**	-	Trivial	30	30	28

Cuadro 5.3: Valores de coverage para los dominios IPC y PPC listados para sus respectivas versiones (trivial, manual y splittings) considerando la heurísticas h^{FF} y LAMA.

- Sin embargo, para los dominios *Freecell* y *Transport* ningún splitting logró mantener el alto coverage que ya tenían los dominios triviales, salvo el caso *Transport* mediante HC con $\gamma = 0.6$ que logró mantener el coverage trivial en 30.

- Con respecto a LAMA, el único splitting que podemos resaltar es HC con $\gamma = 0.8$ para el dominio *Pipesworld* con una mejora del coverage trivial de 38 a 44.

Lo que vimos previamente, evidencia el potencial de nuestra técnica automática, que en muchos casos puede reemplazar efectivamente a un experto en el dominio.

5.5. Splitting sobre dominios ADL

En la sección anterior evaluamos los efectos de la función de splitting, pero, exclusivamente sobre dominios del tipo STRIPS. En este sentido, analizamos su impacto en el rendimiento del planificador: tanto en su proceso de grounding, como de búsqueda. Aún, resta evaluar el impacto de splitting, sobre dominios del tipo ADL, utilizando la extensión de nuestra técnica introducida en la Sección 4.5. Para llevarlo a cabo, consideramos dos dominios de tipo ADL, *Hiking* y *Cavediving*. Primero, nos concentraremos en su efecto en el proceso de grounding del planificador. Para cada dominio, generamos

un conjunto de tareas lo suficientemente complejas para que el dominio, en su versión original, prensente dificultades en el proceso de grounding.

Para el dominio *Hiking* (ver apéndice A.11) generamos 9 tareas. El generador de tareas tiene 3 parámetros: cantidad de parejas, cantidad de vehículos y cantidad de lugares de acampe. De esta manera, la tarea "task_07_08_09" consta de 7 parejas, 8 vehículos y 9 lugares de acampe. En el cuadro 5.4 se muestra la cantidad de acciones generadas por el proceso de grounding, para cada una de las tareas listadas. "None" denota que el proceso de grounding no finalizó con 30 minutos de tiempo ejecución, con 4GB de memoria. Analizando la tabla podemos destacar los siguiente:

- Los tres splittings mostrados HC con $\gamma \in \{0.0, 0.4, 0.6\}$ resuelven el grounding en 3, 4 y 6 tareas sobre 9, respectivamente. Solo 2 tareas fueron resueltas por la versión original. Más aún, en las tareas donde el splitting trivial no finaliza, siempre existe al menos uno que si la resuelve.

- En la tarea "task_07_08_0" el splitting HC con $\gamma = 0.0$ redujo el grounding de 905 mil a 5 mil acciones, aunque los otros dos splitting no llegaron a finalizar.

- En la tarea "task_09_10_11" ningún splitting fue capaz de mejorar el resultado de trivial, inclusive HC con $\gamma \in \{0.0, 0.4\}$ ni siquiera lograron terminar.

- En definitiva, el splitting HC con $\gamma = 0.6$ obtiene el mayor porcentaje (66 %) de tareas que logran finalizar el proceso de grounding, convirtiéndose en la versión de *Hiking* óptima para el conjunto de tareas listadas.

Para el domino *Cavediving* (ver apéndice A.12) generamos 4 tareas. El generador de este dominio también tiene 3 parámetros. En el cuadro 5.5 mostramos el grounding obtenido en las cuatro tareas generadas. Lo que sigue son los resultados salientes del experimento:

- El splitting HC $\gamma = 0.2$ resuelve las 4 tareas mientras que trivial y HC con $\gamma = 0.6$ logran finalizar en 3 de ellas. Además, en la 3 tareas donde trivial finaliza, HC con $\gamma \in \{0.2, 0.6\}$ reducen el grounding en varios ordenes de magnitud.

- En la tarea "task_04_04_04" pasamos de 205 mil acciones a 30 mil si $\gamma = 0.2$ ó 87 mil si $\gamma = 0.6$.

- Lo mismo ocurre para las tareas "task_05_05_05" y "task_06_06_06".

Hiking				
Tareas	Trivial	HC 0.0	HC 0.4	HC 0.6
task_07_08_09	905324	**5560**	None	None
task_08_09_10	None	None	**20604**	230256
task_09_10_11	**3568374**	None	None	389880
task_10_11_15	None	**15370**	None	None
task_11_12_13	None	None	None	**970992**
task_12_13_14	None	None	None	**1449864**
task_13_14_15	None	None	**78459**	2101736
task_14_15_16	None	None	**96618**	None
task_15_16_17	None	**32776**	117379	4104000
Finalizadas	2/9	3/9	4/9	**6/9**

Cuadro 5.4: Grounding obtenido para el dominio *Hiking* utilizando splitting para ADL. Menor grounding resaltado en negrita.

Cavediving			
Tareas	Trivial	HC 0.2	HC 0.6
task_04_04_04	205272	**30417**	87336
task_05_05_05	1127808	**145353**	460080
task_06_06_06	5331648	**643017**	2136288
task_07_07_07	None	**3287241**	None
Finalizadas	3/4	**4/4**	3/4

Cuadro 5.5: Grounding obtenido para el dominio *Cavediving* mediante splitting para ADL. Menor grounding resaltado en negrita.

- Finalmente, se observa que la reducción en el grounding de HC con $\gamma = 0.2$ es siempre superior a la ofrecida por HC con $\gamma = 0.6$. Por lo tanto, HC con $\gamma = 0.2$ se convierte en la versión de *Cavediving* óptima para el conjunto de tareas listadas.

5.6. Validando la función de spliteabilidad

Esta sección intenta dar evidencia empírica para validar la noción de spliteabilidad de la sección 4.6. Recordemos, que spliteabiildad intenta caracterizar aquellos dominios para los cuales la técnica de splitting tiene más chances de producir un efecto positivo sobre el dominio y el planificador. Comenzaremos la evaluación sobre los dominios IPC. En el cuadro 5.6 se muestran 16 dominios IPC (salvo *Pipesworld*) ordenados en forma decreciente según su valor de spliteabilidad. Observamos que, los dominios IPC, no presentan en general un alto valor de splitability, con la excepción del dominio *Freecell*, y en menor medida *Pipesworld*. Notar que esto, explicaría también, los resultados de la Sección 5.3, donde la técnica de splitting, en muchos casos no obtuvo una mejora en el rendimiento de la búsqueda.

Si bien, en estos dominios el grounding es optimizado por el splitting, no resulta ser suficiente para compensar el deterioro en el rendimiento del proceso de búsqueda. No obstante, vamos a evaluar la función de spliteabilidad sobre estos dominios, pero, restringiendo los experimentos a los cinco dominios con mayor valor de spliteabilidad: *Freecell*, *Pipesworld*, *Zenotravel*, *Scanalyzer* y *Hiking*. El experimento consiste en seleccionar el 20 % de los esquemas de un dominio en tres formas diferentes, para luego aplicar la operación de splitting únicamente sobre el 20 % seleccionado. Las formas en que seleccionamos son las siguientes: (1) el 20 % de los esquemas del dominio con mayor valor de spliteabilidad, (2) el 20 % con menor valor de spliteabilidad y (3) el 20 % de los esquemas en forma aleatoria[4].

En lo que resta de la Sección, mostramos el resultado de los experimentos para los cinco dominios mencionados. En cada dominio, el comportamiento esperado es que el grounding producto del splitting sobre (1) sea menor que el grounding en (2) y en (3). Como ya vimos previamente, para la función de splitting, es necesario definir un γ. De las secciones previas, sabemos que el splitting tiene efectos nocivos en la búsqueda, lo cual se incrementa cuando γ es menor. Por esta razón, decidimos utilizar el γ mas cerca de 1, tal que, splitting divide al menos en dos partes todos los esquemas seleccionados en (1), (2) y (3).

Freecell. Este dominio es el que tiene un valor de spliteabilidad mayor de los listados en el cuadro 5.6, con un valor de 0.8. Además, consta de 10 esquemas, cuyos valores de spliteabilidad se muestran en el Cuadro A.8. En particular, para este dominio debemos seleccionar dos esquemas (20 % de 10

[4]Para el caso aleatorio, realizamos tantos sorteos al azar como esquemas de acción tiene el dominio.

	Dominio	**Splty**	**Esq**
1	Freecell	0.80	10
2	Pipesworld	0.62	4
3	Zenotravel	0.57	5
4	Scanalyzer	0.57	4
5	Hiking	0.55	7
6	Cavediving	0.54	8
7	Sokoban	0.53	3
8	Elevators	0.53	6
9	Storage	0.53	5
10	Mystery	0.53	3
11	Transport	0.53	3
12	Floortile	0.48	7
13	Driverlog	0.48	6
14	Logistics	0.48	6
15	Satellite	0.48	5
16	Depot	0.48	5

Cuadro 5.6: Lista de dominios ordenados por su valor de spliteabilidad. "Splty" valor de spliteabilidad; "Esq" cantidad de esquemas de acción.

esquemas) para aplicar la operación de splitting, según el experimento definido. Por lo tanto, se aplica a los esquemas SENDTOFREE-B y NEWCOL-FROMFREECELL, para el conjunto (1). Por otro lado, aplicamos splitting sobre SENDTOHOME y MOVE, para (2), y finalmente realizamos 10 sorteos seleccionando dos esquemas al azar, para (3). El experimento se ejecutó para este dominio con un valor de $\gamma = 0.6$. En el cuadro 5.6 se observa que el splitting de los dos esquemas de mayor spliteabilidad (First) reduce en tres ordenes de magnitud el grounding del dominio original (de 24.4 a 7.6 mil acciones). Para los dos esquemas de menor spliteabilidad (Worst) (3) logramos un resultado un poco peor al dominio trivial (de 24.4 a 25.1 mil acciones). Finalmente, para los dos esquemas al azar (RND) se reduce el grounding de 24.4 a 20.6 mil acciones en promedio. En definitiva, la mayor reducción se observó en First (1), lo que indica que la función de spliteabilidad señala para este dominio, con bastante precisión, aquellos esquemas cuyos splittings tienen más chances de producir una importante optimización del grounding.

Pipesworld. Este dominio es el segundo en valor de spliteabilidad del Cuadro 5.6, con un valor de 0.62. El dominio contiene 4 esquemas con los valores de spliteabilidad que se muestra en el Cuadro A.9. Para este dominio seleccio-

namos 1 esquema (20 % de 4 esquemas) para aplicar la operación de splitting en (1), (2) y (3). Entonces aplicamos splitting al esquemas POP para (1). Luego, aplicamos splitting en POP-UNITARYPIPE, para (2) y finalmente realizamos 4 sorteos seleccionando 1 esquema al azar, para (3). El experimento se ejecutó para este dominio con un valor de $\gamma = 0.6$. Este dominio, tiene la particularidad de que se encuentra disponible una versión de splitting manual, debido a que en su versión original presentaba serias dificultades para completar el procesos de grounding. En particular, los autores del dominio realizaron el splitting manual sobre los esquemas PUSH y POP, dejando el resto de los esquemas sin dividir. Es decir, los esquemas escogidos por los autores para realizar un splitting fueron exactamente los dos esquemas con mayor valor de spliteabilidad. Además, al observar el Cuadro 5.6, el splitting en el esquema con mayor spliteabilidad produce la mayor optimización del grounding. Al dividir el esquema con menor spliteabilidad obtenemos un grounding similar al trivial. Mientras que realizar un splitting a un esquema al azar reduce el grounding promedio (de 691.4 a 622.2 acciones), pero en menor media que First (430).

Zenotravel. Este dominio es el tercero más spliteable entre todos los dominios listados en el cuadro 5.6, con un valor de 0.57. Contiene 5 esquemas, por lo cual, en este dominio debemos seleccionar 1 esquema para el experimento. Entonces aplicamos splitting al esquema ZOOM, para (1). Por otro lado, para (2) se aplica en DEBARK y finalmente realizamos 5 sorteos para (3). El experimento se ejecutó para este dominio con un valor de $\gamma = 0.5$. En el cuadro 5.6 se confirma nuevamente que la mayor optimización del grounding (de 13.4 a 9.3) es aplicando splitting sobre el esquema de mayor spliteabilidad (1). La selección aleatoria reduce el grounding (a 11.5), pero en menor medida. Si aplicamos splitting sobre el esquema de menor spliteabilidad obtenemos casi el mismo grounding que dominio original (11.5).

Scanalyer. Este dominio es el cuarto en el cuadro 5.6, con un valor de 0.57 de spliteabilidad. Contiene 4 esquemas. A igual que en el dominio anterior, debemos seleccionar e1 esquema para aplicar la operación de splitting. Entonces aplicamos splitting al esquema ANALYZE-4, para (1), ANLYZE-2, para (2) y realizamos 4 sorteos para (3). Utilizamos un valor de $\gamma = 0.4$. En el Cuadro 5.6, a diferencia de los anteriores experimentos, observamos que la mayor optimización del grounding (de 87.3 a 82.2) es aplicando splitting aleatoriamente (1). Al aplicar splitting sobre el esquema con mayor spliteabilidad no resultó beneficioso, incrementando el grounding de 87.3 a 94.1. Mientras que aplicar splitting sobre el esquema de menor spliteabilidad es

Dominio	Trivial	Best	Worst	RND
Freecell	24.4	**7.6**	25.1	20.6
Pipesworld	691.4	**430.0**	691.7	622.2
Zenotravel	13.4	**9.3**	13.4	11.5
Scanalyzer	87.3	94.1	87.3	**82.8**
Hiking	882.2	**207.1**	880.5	703.4

Cuadro 5.7: Cantidad de acciones generadas por el proceso de grounding (expresadas en unidades de mil). "Best"(1) grounding correspondiente al splitting sobre esquemas con mejor valor de spliteabilidad; "Worst"(2) grounding correspondiente al splitting sobre esquemas con peor valor de spliteabilidad; "RND"(3) grounding correspondiente al splitting sobre esquemas seleccionados al azar. El mejor valor de grounding se resalta en negrita.

indiferente porque no afecta el grounding del dominio original.

Hiking. Finalmente, este es el último dominio en el cuadro 5.6, con un valor de 0.55. Contiene 7 esquemas, por tanto, debemos seleccionar 2 esquemas para ejecutar cada experimento. Entonces aplicamos splitting sobre los esquemas DRIVE_TENT_PASSENGER y DRIVE_TENT, para (1), PUT_DOWN y PUT_UP para (2) y finalmente realizamos 7 sorteos seleccionando dos esquemas al azar. El experimento se ejecutó para este dominio con un valor de $\gamma = 0.5$. Los resultados obtenidos en el cuadro 5.6, evidencian que la reducción del grounding es sorprendente, alrededor de cuatro ordenes de magnitud (de 882.2 a 207.1) cuando dividimos los dos esquemas con mayor spliteabilidad. Cuando hacemos el splitting de los dos esquemas de menor spliteabilidad, no se produce una reducción del grounding significativa (de 882.2 a 880.5). Mientras que si dividimos dos esquemas al azar, la reducción promedio es mayor (de 882.2 a 703.4), pero, muy lejos de la alcanzada por los dos esquemas utilizados en (1). En definitiva, en este dominio nuevamente el valor de spliteabilidad indica con precisión aquellos esquemas que más influyen en el grounding de las tareas, tal como ocurrió en los tres primeros dominios ya vistos.

En conclusión, podemos decir que en cuatro de los cinco dominios evaluados en esta oportunidad, la noción de spliteabilidad tiene éxito en señalar aquellos con mayor oportunidad de reducir su grounding. En otras palabras, a mayor valor de spliteabilidad de un dominio, mayor es la necesidad del dominio de ser optimizado vía la función splitting.

Capítulo 6

Conclusiones

En este trabajo hemos logrado automatizar una idea que ya existía previamente en la comunidad de planning pero que nunca antes había sido formalmente estudiada. La idea consiste en partir los distintos esquemas de acción de un dominio en sub-esquemas, de forma tal que cada sub-acción ejecuta una parte de la acción y la ejecución de todas las sub-acciones produce exactamente el mismo efecto sobre el entorno que la ejecución de la acción. Con esto conseguimos reducir el tamaño de las interfaces de los esquemas que presenta un dominio con la expectativa de que esto se traduzca en una importante reducción en la cantidad de acciones proposicionales generadas por el proceso de grounding del planificador. Splitting constituye un trade-off entre la reducción del tamaño de las interfaces (por ende, la disminución de la cantidad de acciones proposicionales) y la cantidad de sub-acciones (por ende, el aumento de la distancia a la meta). Esto convierte a la técnica de splitting en una herramienta muy útil para optimizar el nivel de granularidad óptimo en las acciones de un dominio.

En el capítulo 4 introducimos los conceptos fundamentales asociados a la técnica. Comenzamos introduciendo un ejemplo que ilustra perfectamente las ventajas de un splitting pero también sus dificultades inherentes. Demostrando que la operación no es trivial y que implica una ingeniería importante para su correcta aplicación. Para suplir dichas dificultades, en la sección 4.2 presentamos el concepto de validez de un splitting y secuencializaciones correctas. Esto junto a la decoración de sub-esquemas desarrollada en la sección 4.3, la cual, fue implementada como un sistema de tokens entre las sub-acciones, aseguran una relación uno a uno entre los planes del dominio original y el dominio producto de la operación de splitting. En definitiva, las secciones anteriores introducen los conceptos necesarios para garantizar que la técnica de splitting sea correcta y completa. En la sección 4.4 estudiamos como computar un splitting a partir de un esquema de acción. Demostramos

su complejidad computacional cuando es formulado como un problema de decisión (NP-completo) y dimos un método, conocido como Hill Climbing, para aproximar la solución en forma general. Este método obtiene una solución que se corresponde con una solución local que optimiza el trade-off ya mencionado. En la sección 4.5 presentamos la extensión de la técnica a dominios de tipo ADL, reutilizando la mayoría de los conceptos introducidos para dominios de tipo STRIPS. Con la simple introducción del concepto de black-box y una redefinición de esquema de acción en términos de black-boxes, resultó suficiente para poder reutilizar todo el framework desarrollado en las secciones anteriores para el caso de esquemas STRIPS. En la sección 4.6 presentamos la noción de spliteabilidad que surgió como un primer intento por caracterizar aquellos dominios donde la optimización vía splitting puede llegar a funcionar mejor. Este concepto ofrece una medida de cuan relacionadas entre sí se encuentran las variables que componen la interfaz de un esquema de acción a través de los distintos átomos que ocurren en la precondición y/o efecto del mismo. Para ello, ponderamos atributos tales como tamaño de la interfaz, cantidad de componentes conexas y densidad del grafo de spliteabilidad asociado al esquema. Esta ponderación se convierte en una métrica del nivel de "spliteabilidad" que posee un esquema. Por supuesto, el valor de spliteabilidad de un esquema es directamente proporcional a las chances de que splitting produzca efectivamente una optimización del mismo.

En el capítulo 5 presentamos los resultados obtenidos de los diferentes experimentos realizados para evaluar cada uno de los conceptos vistos en el capítulo 4. Entonces, comenzamos con una evaluación del framework para aplicar splitting sobre dominios STRIPS. Nos enfocamos primero en observar el efecto del splitting sobre la representación PDDL del domino, concluyendo que el splitting logra muchas veces encontrar un muy buen trade-off entre la reducción de las interfaces de los esquemas y la cantidad de sub-esquemas resultantes. Después, analizamos el impacto del splitting sobre el proceso de grounding del planificador, concluyendo que splitting siempre reduce la cantidad de acciones generadas por el grounding con respecto al dominio original, salvo casos correspondientes a splittings muy agresivos producto de valores de γ cercano a cero. Incluso, en muchos casos la reducción del grounding fue de hasta dos órdenes de magnitud principalmente en dominios que no eran IPC como *Pipesworld*. Luego, analizamos el efecto del splitting sobre el proceso de búsqueda y advertimos un aumento considerable de los estados expandidos que se explica al haber aumentado la distancia a la meta como producto del reemplazo de acciones por sub-acciones. Surge el interrogante de si este fenómeno se produjo por la introducción de estados intermedios producto de la ejecución parcial de los splitting ó si se produjo directamente como consecuencia de un cambio en los valores heurísticos de los distintos

estados. Quizás podamos informar al algoritmo de búsqueda que se trata de una búsqueda sobre un "espacio spliteado" y de esa forma optimizar la cantidad de veces que se llama a la heurística durante el proceso de búsqueda. Por todo esto, concluimos para el caso de dominios del tipo STRIPS, que la operación de splitting debe ser aplicada delicadamente, en el sentido de que debe partir los esquemas lo menos posible para evitar introducir muchas sub-acciones pero al mismo tiempo disminuir la interfaz de los esquemas lo más posible. Luego, realizamos algunos experimentos para evaluar el rendimiento de la extensión del splitting a dominios ADL. Para ello utilizamos los dominios *Cavediving* y *Hiking* aunque en esta oportunidad nos enfocamos exclusivamente en analizar el efecto del splitting en el grounding obtenido. En ambos dominios observamos que el splitting redujo considerablemente el grounding con respecto a la versión original de los dominios, evidenciando que el comportamiento de la técnica parece ser prometedora con respecto a dominios ADL. Por último, realizamos ciertos experimentos con el propósito de evaluar la noción de spliteabilidad en aquellos dominios que presentaban un alto valor del mismo. Observamos que el splitting producto de dividir solamente aquellos esquemas del dominio con mayor valor de spliteabilidad termina generando una mayor reducción del grounding que si dividimos aquellos esquemas con peor valor de spliteabilidad o incluso si dividimos esquemas seleccionados al azar. Por lo tanto, concluimos que la noción de spliteabilidad propuesta tiende a caracterizar con bastante precisión aquellos esquemas y dominios que realmente necesitan ser optimizados vía splitting.

En definitiva, dados los fundamentos de la técnica y dados los resultados observados en los distintos experimentos realizados, podemos asegurar que nuestro método de optimización por splitting resulta ser una herramienta muy útil y prometedora para afrontar el problema de grounding principalmente en aquellos dominios que se caracterizan por tener largas interfaces en sus esquemas de acción y tareas cuyo tamaño proposicional hace inviable el proceso de grounding del planificador dado sus recursos computacionales. Esto se justifica en el hecho de que estos dominios vía splitting pueden sufrir una reducción lo suficientemente importante en la cantidad de acciones generadas por el proceso de grounding como para contrarrestar el incremento en el factor de profundidad del espacio de búsqueda producto del reemplazo de acciones por sub-acciones.

Parte III

Subdominización

Capítulo 7

Fundamentos de Subdominización

La mayoría de los planificadores modernos realizan una etapa de preprocesamiento donde computan una representación proposicional (de más bajo nivel) de la tarea, sobre la cual finalmente realizan la búsqueda, a partir de una especificación esquemática de más alto nivel. La representación proposicional resulta ser la más adecuada para la aplicación directa de efectivas técnicas de búsqueda heurística. Mientras que la representación esquemática resulta más conveniente para el usuario que modela el problema, ya que, permite capturar en forma flexible y compacta propiedades generales del problema a modelar. Entonces, resulta de vital importancia la existencia de una componente dentro del planificador que realice esta transformación o compilación en forma eficiente. Esta componente recibe el nombre de *proceso de grounding* y es el objeto de estudio de este capítulo. En particular, en este trabajo proponemos una optimización del proceso de grounding por alcanzabilidad relajada (ver sección 3.4) implementado actualmente en el planificador FD [Helmert, 2006] por ser este el sistema actualmente más utilizado por la comunidad de planning.

En la sección 3.2 del capítulo 3 mencionamos que el tamaño de la representación proposicional de una problema es exponencial en la aridad de los predicados y los esquemas de acción del dominio. Cuando el dominio está fijo, el tamaño de la representación proposicional es polinomial en la cantidad de objetos definidos en el problema. Sin embargo, esto último todavía puede ser prohibitivo cuando el número de objetos definidos es demasiado grande. Por lo tanto, en la práctica la representación proposicional de una tarea puede agotar con bastante facilidad los recursos computacionales disponibles del planificador, ya sea, tiempo y/o memoria, causando que este último falle aún sin comenzar la etapa de búsqueda. Para intentar solucionar esta dificultad,

presentamos una técnica que consiste en generar sólo una proyección de la tarea proposicional que contenga al menos una solución (plan) de la misma. Concretamente, la técnica computa un subconjunto de las acciones proposicionales, llamado sub-dominio, que contiene las acciones más relevantes para resolver la tarea en cuestión. Por ejemplo, un sub-dominio ideal consiste en generar sólo las acciones que pertenecen a un determinado plan. Pero esto es tan difícil como computar un plan de la tarea y sabemos que esto es PSPACE-completo. Por lo tanto, en la práctica tenemos un trade-off entre el tamaño del sub-dominio y su probabilidad de contener una solución. En particular, la técnica que proponemos en este capítulo computa sub-dominios utilizando técnicas de aprendizaje automático con el fin de identificar aquellas partes de la tarea que pueden ser más relevantes para su resolución. Los resultados empíricos demuestran que este método puede ser capaz de resolver tareas de planning donde su representación proposicional completa produce la falla del planificador. Convirtiendo a nuestra técnica en una herramienta muy valiosa para afrontar el problema de grounding principalmente en aquellos dominios que presentan esta dificultad.

7.1. Grounding Parcial

Mejorar el proceso de grounding que lleva adelante un planificador es un desafío abierto y de vital importancia en aquellas tareas de planning en donde sólo una parte de esta es realmente relevante para alcanzar la meta. Por ejemplo, en el dominio *Robotics*, la tarea proposicional contiene todos lo objetos con los cuales el robot puede llegar a interactuar, incluso aquellos que no son necesarios para resolver una tarea determinada [Lang and Toussaint, 2009]. En dominios que modelan entornos de ciber-seguridad, como es el caso del dominio *Caldera* [Miller *et al.*, 2018], una tarea puede contener información sobre absolutamente todos los detalles de la red y muchos de ellos pueden ser descartados a la hora de resolver la tarea. En la actualidad, no se conoce en la literatura de planning ningún método de grounding que intente focalizar este proceso , desde el inicio, en aquellas partes de la tarea que son más relevantes para su resolución.

A partir de esta sección introducimos la idea de grounding parcial, donde en lugar de generar todas las acciones relajadamente alcanzables de una tarea de planning (ver sección 3.4) y el cual es referido a partir de este momento como *"grounding total"*, intentamos generar solamente aquellas acciones que resultan verdaderamente necesarias para alcanzar la meta. El algoritmo 4 muestra el proceso de grounding total implementado actualmente por el planificador Fast Downward [Helmert, 2006]. Este realiza una computación

de punto fijo muy similar a la computación del grafo de planning relajado [Blum and Furst, 1997]. En este algoritmo una cola q es inicializada con los hechos (o "facts", en inglés) del estado inicial. En cada iteración un elemento de la cola es extraído y marcado como procesado. Si el elemento extraído es un fact, entonces aquellas acciones aún no procesadas, que son aplicables dados los facts procesados hasta ese momento, son agregadas a la cola. Si el elemento extraído es una acción, entonces todos los facts áun no procesados que ocurren en la lista *add* de la acción, son colocados en la cola. El algoritmo termina cuando la cola se vacía. De esta forma, todos los facts y acciones que fueron procesados son exactamente aquellos relajadamente alcanzables desde el estado inicial. Cabe aclarar que por simplicidad el algoritmo que se describe solo considera tareas STRIPS pero puede ser adaptado para soportar atributos PDDL tales como precondiciones negativas, o inclusive efectos condicionales como es descripto en [Helmert, 2009].

Algorithm 4: Algoritmo de Grounding Total implementado actualmente en el planificador Fast Downward (FD).

Entrada: s_0 estado inicial de la tarea.
Salida: A y F conjunto de acciones procesadas y facts, respectivamente.

```
1:  q := s_0
2:  F := ∅ //facts procesados
3:  A := ∅ //acciones procesadas
4:  while ¬q.isEmpty() do
5:     e := q.pop()
6:     if e.isFact() ∧ e ∉ F then
7:        F := F ∪ {e}
8:        for each a ∉ A ∧ pre(a) ⊆ F do
9:           q.insert(a)
10:       end for
11:    end if
12:    if e.isAction() ∧ e ∉ A then
13:       A := A ∪ {e}
14:       for each f ∈ add(a) ∧ f ∉ F do
15:          q.insert(f)
16:       end for
17:    end if
18: end while
19: return A, F
```

En el algoritmo de grounding total es indiferente el orden en que se pro-

cesan los elementos de la cola, pues al finalizar el proceso todos las acciones relajadamente alcanzables terminan tarde o temprano siendo procesadas. Esto es producto directo de la condición de terminación del algoritmo, es decir, que la cola se encuentre vacía. Sin embargo, si logramos procesar primero aquellas acciones que son realmente las más relevantes para resolver la tarea, entonces no es necesario esperar a que la cola se vacíe y esto permite finalizar el algoritmo antes. De esta manera, generamos una menor cantidad de acciones proposicionales, ya que, las acciones sin relevancia nunca llegaron a ser procesadas por el algoritmo. Intuitivamente, esto parece ser una buena idea, ya que, la mayoría de las tareas de planning se caracterizan por tener planes cortos (a lo sumo centenares de acciones) en comparación a la cantidad de acciones proposicionales que se obtiene producto del proceso de grounding total (del orden de millones de acciones usualmente). En definitiva, si logramos detectar aquellas acciones verdaderamente relevantes de una tarea de planning dada, esto posibilita un ahorro muy importante en los recursos computaciones (tiempo y memoria) disponibles para el proceso de grounding del planificador. Así surge la idea de grounding parcial que consiste básicamente en dos modificaciones en el algoritmo de grounding total: 1) modificamos el orden en que las acciones son procesadas; y 2) modificamos la condición de terminación, es decir, el algoritmo puede detenerse antes que la cola se encuentre vacía. Para estas dos modificaciones, vamos a proponer alternativas con el objetivo de minimizar la cantidad de acciones que resultan del proceso de grounding y al mismo tiempo preservar al menos una solución de la tarea.

La técnica de grounding parcial es implementada en el algoritmo 5. Los principales cambios con respecto al algoritmo de grounding total son: el algoritmo procesa no más de N acciones y además son procesadas según su nivel de relevancia en forma decreciente. Es muy importante observar que debido a que el algoritmo puede terminar antes que la cola se vacíe, estamos obligados a procesar la totalidad de los facts existentes en la cola antes de poder procesar cualquier acción, es decir, el algoritmo otorga preferencia en la cola a los facts por sobre las acciones. Esto garantiza que todos los facts que se encuentran en los efectos de las acciones ya procesadas están en F. Así, evitamos indefiniciones que podrían surgir como consecuencia de que un fact en los efectos de una acción no haya sido procesado aún por el algoritmo. Esto último, se puede generalizar por medio de la siguiente definición:

Definición 43 (Fact-Consistencia) *Sea $\mathcal{A}$ un algoritmo de grounding y sea F, A el conjunto de facts y acciones retornadas por $\mathcal{A}$, respectivamente. Decimos que $\mathcal{A}$ es "fact-consistente" si para todo $a \in A$, se cumple que, si $p \in pre(a) \cup del(a) \cup add(a)$, entonces $p \in F$.*

Algorithm 5: Algoritmo de Grounding Parcial.

Entrada: s_0 estado inicial de la tarea, R función de relevancia y $N \in \mathbb{N}$.
Salida: A y F conjunto de acciones procesadas y facts, respectivamente.

```
1:  q := s_0
2:  F := ∅ //facts procesados
3:  A := ∅ //acciones procesadas
4:  while ¬q.isEmpty() ∧ |A| ≤ N do
5:    if q.containsFact() then
6:      f := q.popFact()
7:      F := F ∪ {f}
8:      for each a ∉ A ∧ pre(a) ⊆ F do
9:        q.insert(a)
10:     end for
11:   else
12:     a := q.popMaxRelevanceAction(R)
13:     A := A ∪ {a}
14:     for each f ∈ add(a) ∧ f ∉ F do
15:       q.insert(f)
16:     end for
17:   end if
18: end while
19: return A, F
```

Un algoritmo de grounding fact-consistente evita tener acciones con precondiciones y/o efectos indefinidos. Claramente, los algoritmos de grounding total y parcial mostrados son fact-consistente. El primero de ellos, trivialmente por definición de alcanzabilidad relajada y el segundo debido a la preferencia de facts por sobre acciones en la cola. Esta propiedad será nuevamente mencionada más adelante cuando estudiemos el concepto de reglas condicionales en la sección 7.3.

En definitiva, sólo después de que todos los facts en la cola han sido procesados, el algoritmo de grounding parcial selecciona una acción de la cola de acuerdo a su relevancia estimada por la función R. La relevancia de una acción está directamente relacionada con la probabilidad de que esta pertenezca a un plan de la tarea. Para estimar estas probabilidades, usamos técnicas de maching learning (ML). Adicionalmente, implementamos cierta *diversidad* entre las acciones seleccionadas, con el fin de evitar un posible *bias* hacia algún esquema particular del dominio que puede llegar a tener el estimador de relevancia. Para ello, consideramos adicionalmente un criterio

round robin (RR) que selecciona acciones de diferentes esquemas en cada iteración. RR funciona en combinación con la función de relevancia, donde esta última decide que acción puntual de un esquema debe ser procesada a continuación.

Es importante destacar que el algoritmo de grounding parcial es *correcto*: si hallamos un plan de la tarea obtenida por grounding parcial, entonces ese mismo plan es también un plan de la tarea que se obtendría por grounding total. Pero desafortunadamente, el algoritmo no es *completo*, es decir, una tarea obtenida por grounding parcial tiene solución si y sólo si las acciones correspondientes de al menos un plan (de la tarea obtenida por grounding total) fueron procesadas. En las próximas secciones nos enfocaremos en estudiar funciones de relevancia, condiciones de parada, o sea, cómo asignar un valor al parámetro N del algoritmo y finalmente como sobrellevar el problema de su incompletitud.

7.1.1. Acciones Relevantes

El éxito de la técnica de grounding parcial depende fundamentalmente del orden en que las acciones de la cola son procesadas, es decir, el proceso de grounding resulta más eficiente si logramos procesar solamente las acciones justas y necesarias para alcanzar la meta. Por lo tanto, la idea consiste en procesar los elementos de la cola de acuerdo a su contribución para alcanzar la meta. Este valor se obtiene utilizando métodos conocidos de ML que permiten obtener un estimador que dada una acción cualquiera estima su probabilidad de pertenecer a un plan óptimo. Observar que exigimos que una acción sea considerada relevante si esta pertenece a un plan óptimo de la tarea y no simplemente a un plan cualquiera. Esta decisión está justificada por el hecho de que ciertas acciones de un plan no óptimo pueden ser eliminadas del mismo y la secuencia de acciones resultante continúa siendo un plan, por lo tanto, esas acciones eliminadas no eran realmente relevantes para alcanzar la meta a pesar de que formaban parte del plan. En cambio, si el plan es óptimo, esto nos asegura que todas sus acciones son relevantes, ya que, si eliminamos cualquiera de ellas, la secuencia resultante ya no es un plan de la tarea.

Más formalmente, dado una tarea de planning $\Pi = (F, A, s_0, g)$ queremos estimar la función de relevancia perfecta $R^*_\Pi : A \to \{0, 1\}$ tal que:

$$R^*_\Pi(a) = \begin{cases} 1 & \textit{si } a \textit{ pertence a un plan óptimo de } \Pi \\ 0 & \textit{caso contrario} \end{cases} \tag{7.1}$$

Para diferentes modelos de ML obtendremos una función de relevancia $R_\Pi : A \to [0, 1]$ que dada una acción estima la probabilidad de que esta pertenezca a un plan óptimo de la tarea. La etapa de aprendizaje se lleva a acabo

utilizando un conjunto de tareas de entrenamiento donde el planificador termina en forma exitosa, es decir, el algoritmo de grounding total termina pudiendo construir la representación proposicional de la tarea y además el proceso de búsqueda heurística sobre esta representación finaliza devolviendo un plan óptimo[1]. De esta manera, logramos construir un conjunto de entrenamiento que consiste de una lista de acciones proposicionales donde a cada una de ellas se le asocia un bit 1 o 0 como etiqueta que indica si la acción ocurrió en el plan óptimo devuelto por el planificador para su respectiva tarea. Así, conseguimos formular el problema de obtener una función de relevancia como un problema de clasificación o regresión (supervisada) donde se estima la probabilidad de pertenecer a la clase 1.

Sin embargo, aún resta transformar las acciones proposicionales en un vector de atributos que permita abstraer las acciones proposicionales de forma tal de poder generalizar independientemente de los objetos concretos del conjunto de entrenamiento. En otras palabras, como el conjunto de entrenamiento y el conjunto de evaluación tendrán diferentes objetos en su definición, entonces dichos atributos no deben referir a objetos concretos. Por lo tanto, el aprendizaje debe ser hecho a nivel esquemas de acción y este trabajo es sin duda la parte más complicada de toda la técnica. Para ello, proponemos que los atributos sean reglas relacionales, las cuales, intentan capturar una relación o conexión entre las diferentes instanciaciones posibles de un esquema de acción con el estado inicial y/o meta de la tarea. A su vez, debido a que los esquemas de acción de un dominio tienen diferentes tipo y cantidad de parámetros, entonces los atributos que caracterizan a cada esquema deben ser también diferentes. Por lo tanto, aprendemos un modelo separado para cada esquema $a[X]$ en un dominio dado. Al final, como todos estos modelos predicen lo mismo, es decir, la probabilidad de que una acción instanciada (de un determinado esquema) sea parte de un plan óptimo, entonces los valores provenientes de diferentes modelos son perfectamente comparables entre si.

Nuestros reglas relacionales fueron diseñadas con el propósito de contar con atributos adecuados para los algoritmos estándares de clasificación-regresión [Kramer *et al.*, 2001]. Un regla consiste de un encabezado y un cuerpo. El encabezado se corresponde con el nombre de un esquema de acción y su interfaz. El cuerpo consiste en un conjunto de predicados atómicos clasificados como de inicio o de meta. Las variables de estos predicados son instanciadas a las variables que ocurren en el encabezado de la regla cuando el tipo de la variables en cuestión lo permite, en caso contrario la variable se

[1]En particular, se resolvieron las tareas de entrenamiento con una búsqueda BFS simbólica bidireccional [Torralba *et al.*, 2017].

considera libre.

Ejemplo 3 *Regla relacional generada para el esquema de acción turn-to(s, to, from) del dominio Satellite. El encabezado de la regla consiste en el nombre del esquema de acción y su interfaz. Mientras que el cuerpo cuenta con dos predicados meta y tres predicados iniciales. El predicado meta have-image(to, modo1) tiene la variable ligada "to" y la variable libre "modo*1*". Más abajo, se muestra también la definición del esquema de acción involucrado en la regla.*

```
turn-to[s,from,to] := goal:have-image[to,modo1],
    goal:have-image[from,modo2],ini:on-board[intrument,s]
    ini:supports[intrument,modo1],ini:supports[intrument,modo2]

(:action turn_to
   :parameters (s:satellite from:direction to:direction)
   :precondition (and (pointing s from) (not (= to from)))
   :effect (and  (pointing s to)(not (pointing s from)))
)
```

Una regla es evaluada para una acción reemplazando las variables del encabezado por los objetos que fueron utilizados para instanciar la acción y chequeando si existe una asignación de las variables libres de los predicados en el cuerpo de la regla, tal que, los facts que se originan ocurren en el estado inicial o en la meta de la tarea. Si esto ocurre, decimos que la acción cumplió la regla. De esta manera, construimos un vector de atributos binarios para cada acción instanciada del conjunto de entrenamiento, donde cada componente del vector indica si la acción correspondiente cumple (1) o no (0) con la regla asociada. Esto resulta en un conjunto de entrenamiento donde cada acción instanciada es representada como un vector de atributos binarios, donde la ultima componente es la clase a la que pertenece. Sobre este conjunto de entrenamiento, utilizamos métodos de clasificación o regresión para obtener el modelo que mejor se ajusta a estos datos y que será utilizado para predecir la clase de nuevas acciones instanciadas de diferentes tareas. Es decir, durante el proceso de grounding parcial, para cada acción que es insertada en la cola, evaluamos las reglas según su esquema de acción y obtenemos su vector de atributos binario, el cual es pasado como entrada al modelo estimador y este devuelve finalmente la relevancia estimada de la acción.

Aclaramos que la mayoría de las reglas generadas no proporcionan información útil para predecir si una acción pertenece a un plan óptimo. Esto es consecuencia directa de utilizar fuerza bruta para generar todas las posibles

reglas. Es decir, carecemos de un mecanismo para determinar de antemano cuáles son aquellas reglas determinantes que logran capturar el concepto de relevancia. Por lo tanto, generamos las reglas en forma exhaustiva y masiva. Y luego de generarlas seleccionamos aquellas reglas que brindan mayor ganancia de información en dos etapas. Primero removemos todas la reglas que se evalúan al mismo valor (0 ó 1) en todas las instancias de entrenamiento. Y segundo, empleamos un método de selección de atributos (feature selection) para descartar aquellas reglas que no aportan ganancia de información a la hora de predecir relevancia. También es importante aclarar que puede ocurrir que un mismo vector de atributos binarios tenga una clase distinta como etiqueta. En este caso, reemplazamos todos esos vectores de entrenamiento iguales por un único vector de ellos, pero con la clase asignada en 1 si al menos uno ellos es clasificado de esta forma, 0 en caso contrario.

7.1.2. Condiciones de Parada

En el algoritmo de grounding total o por alcanzabilidad relajada el proceso se detiene cuando la cola se vacía. Esto implica que no importa el orden en que las acciones y facts son procesados, ya que, de todos modos serán finalmente procesados. Sin embargo, el contar con un modelo que estima relevancia, posibilita detener el algoritmo antes que la cola se vacíe como consecuencia de que el algoritmo puede procesar exclusivamente aquellas acciones con mayor relevancia, dejando sin procesar al resto. Claramente, mientras más preciso sea el estimador de relevancia, menor será la cantidad de acciones que debemos procesar para contener un plan de la tarea, permitiendo de esta manera finalizar antes el proceso de grounding. En definitiva, si el estimador de relevancia selecciona las acciones verdaderamente relevantes, en particular, aquellas que pertenecen a un plan óptimo de la tarea, entonces grounding parcial puede finalizar mucho antes que el proceso de grounding total. Surge el interrogante de cómo determinar la cantidad de acciones que se deben procesar (en el orden determinado por la función de relevancia estimada) para contener una plan de la tarea. Para comenzar a responder esto, consideremos las siguientes dos afirmaciones:

1. Una tarea parcialmente procesada tiene un plan relajado si y sólo si los facts de la meta han sido procesados ($g \subseteq F$).

2. Una tarea parcialmente procesada tiene un plan si y sólo si sus acciones contienen las acciones correspondientes a un plan de la tarea totalmente procesada.

La afirmación 1 proporciona una condición mínima necesaria para que la tarea parcialmente procesada pueda contener un plan, ya que, sabemos que si la tarea relajada no tiene un plan, entonces la tarea (sin relajar) tampoco lo tiene. Por lo tanto, el proceso de grounding parcial debe continuar al menos hasta que se cumple la condición $g \subseteq F$. No obstante, el cumplimiento de esta condición no es suficiente para asegurar que la tarea parcialmente procesada tenga efectivamente un plan. La afirmación 2 proporciona una condición obvia de parada donde la tarea parcialmente procesada contiene un plan de la tarea totalmente procesada pero difícil de predeterminar. En esta sección vamos a proponer diferentes condiciones de parada. Todas con el objetivo de maximizar la probabilidad de que las acciones de una tarea parcialmente procesada contiene un plan de la tarea totalmente procesada.

Definición 44 *Sea $R : A \to [0, 1]$ una función de relevancia estimada, entonces definimos GG^+_R (Relaxed Goal Grounding) como la cantidad de acciones procesadas por el algoritmo, en el orden determinado por R, hasta que $g \subseteq F$. Y definimos, RG^+_R (Relaxed Reachability Grounding) como la cantidad de acciones procesadas por el algoritmo, en el orden determinado por R, hasta que la cola se vacía.*

El valor RG^+_R es exactamente la cantidad de acciones relajadamente alcanzables y recordar que estas son independientes del orden en que fueron procesadas por el algoritmo, es decir, $RG^+_R = RG^+_{R'}$ para cualquier función de relevancia estimada R, R'. Así, podemos obviar el subíndice que hace referencia a la función de relevancia. El valor GG^+_R depende de la función de relevancia estimada R, por lo tanto, el subíndice será obviado solamente si el contexto determina sin ambigüedad a que función de relevancia nos referimos. Hechas estas aclaraciones, determinar una condición de parada es equivalente a resolver el siguiente problema:

Definición 45 *Dada una función de relevancia estimada R, hallar un valor $N \in \mathbb{N}$ de acciones a procesar, en el orden determinado por R, tal que, $GG^+ \leq N \leq RG^+$ y que las acciones resultantes contengan un plan de la tarea totalmente procesada.*

Para tratar con este problema proponemos dos estrategias:

- *Estrategia 1*: consiste en procesar tantas acciones como sea posible pero restringido a los recursos computacionales (tiempo y memoria) que posee el planificador para llevar adelante el proceso de grounding. Es decir, continuar procesando acciones mientras dichos recursos computacionales no se vean comprometidos. Entonces, sea M una constante, provista

como parámetro por el usuario, que representa la cantidad máxima de acciones que se pueden procesar dado los recursos computacionales disponibles, entonces el algoritmo continua procesando acciones siempre que $|A| \leq M$. El caso interesante se presenta cuando $GG^+ \leq M \leq RG^+$ porque en caso contrario las acciones procesadas no pueden contener una solución de la tarea (si $M < GG^+$) o contienen una solución pero trivialmente (si $RG^+ < M$). La debilidad de esta estrategia radica en que es complicado para el usuario de la técnica asignar un valor preciso al parámetro M. Resulta difícil conocer de antemano la cantidad de acciones máximas que se pueden procesar dada una cierta cantidad de tiempo y memoria disponible.

- *Estrategia 2*: consiste en utilizar nuevamente ML con el fin de estimar la cantidad de acciones suficientes a procesar para contener un plan de la tarea, dada una función de relevancia estimada R. Este valor lo denotamos con SG_R (sufficient grounding) y también obviaremos el sub-índice que hace referencia a su función de relevancia estimada siempre que el contexto lo permita. Finalmente, el algoritmo continua procesando acciones siempre que $|A| \leq SG$.

Para aplicar la estrategia 2 necesitamos construir un nuevo conjunto de entrenamiento donde una instancia es de la forma $t_i = (GG^+, SG, RG^+)$ con t_i la i-ésima tarea de entrenamiento seleccionada. Para la tarea t_i, el valor GG^+ se obtiene corriendo el algoritmo hasta que $g \subseteq F$. Similarmente, el valor RG^+ se obtiene corriendo el algoritmo hasta que la cola se vacía. Esto implica, que las tareas de entrenamiento t_i son seleccionadas de forma tal que el proceso de grounding total termina exitosamente. Por último, el valor de SG se obtuvo corriendo el algoritmo (posiblemente en varias oportunidades) utilizando la estrategia 1 de la siguiente manera: asignamos $M = 10$ mill y chequeamos si con ello es suficiente para contener un plan de la tarea. En caso afirmativo, terminamos y asignamos $SG = M$. En caso negativo, asignamos $M = M + 10k$ y corremos de nuevo el algoritmo. Así sucesivamente hasta hallar un valor para SG. Entonces, dado un conjunto de entrenamiento $\{t_i\}$, proponemos como modelo a entrenar para predecir el valor de SG la siguiente ecuación lineal:

$$SG = GG^+ + a.(RG^+ - GG^+) \; con \; a \in [0, 1] \tag{7.2}$$

Observar que la definición del modelo propuesto, asegura que el valor de SG es consistente, es decir, siempre satisface la desigualdad $GG^+ \leq SG \leq RG^+$. Si en el conjunto de entrenamiento se observa que el valor de SG tiende a estar cerca de GG^+, entonces eso hará que el coeficiente a del modelo tienda

a 0. Mientras que si en el conjunto de entrenamiento ocurre que SG tiende a estar cerca del valor de RG, entonces el parámetro a tiende a 1. Es decir, el entrenamiento consiste simplemente en hallar el valor de a que mejor ajusta al conjunto de entrenamiento. Una dificultad de esta estrategia es que para tareas nuevas o de evaluación, no podemos asumir que el valor de RG^+ se conoce, ya que, justamente las tareas de evaluación interesantes son aquellas en las cuales el proceso de grounding total falla. Para suplir esta dificultad, proponemos estimar el valor de RG^+, mediante ML, a partir de atributos extraídos directamente de las descripción PDDL de la tarea. Consideramos los siguientes atributos:

- obj: cantidad de objetos definidos
- $init$: cantidad de facts que ocurren en el estado inicial
- $goal$: cantidad de facts que ocurren en la meta
- GG^+: cantidad de acciones procesadas hasta alcanzar relajadamente la meta.

Entonces la instancia de entrenamiento correspondiente a la tarea t_i es de la forma $t_i = (obj, init, goal, GG^+)$ y utilizamos la siguiente regresión lineal para estimar el valor de RG^+:

$$RG^+ = a_1.obj + a_2.init + a_3.goal + a_4.GG^+ \tag{7.3}$$

El entrenamiento consiste en hallar valores para los coeficientes a_i que mejorar ajustan a las instancias de entrenamiento. Para una tarea nueva o de evaluación calculamos un valor estimado de RG^+ mediante la ecuación 7.3 y luego este valor es utilizado en la ecuación 7.2 para estimar el valor de SG.

7.1.3. El Problema de la Verificación

Una vez que contamos con un estimador de relevancia y una cantidad determinada de acciones a procesar, la tarea parcialmente procesada que se obtuvo debe ser verificada, es decir, chequear si esta contiene efectivamente un plan de la tarea. Esta verificación indefectiblemente nos lleva a ejecutar el proceso de búsqueda del planificador sobre la tarea parcialmente procesada. Por lo tanto, la verificación está restringida también a la cantidad de tiempo y memoria que disponemos para realizar esta búsqueda, la cual, puede arrojar tres resultados posibles:

1. Hay plan, por tanto, el proceso de grounding parcial fue exitoso.

2. No hay plan, por lo tanto, el proceso de grounding parcial fracasó. [2]

3. La búsqueda falló producto de agotar los recursos computacionales disponibles, por lo tanto, en este caso no podemos determinar si el proceso de grounding parcial fue exitoso o fracasó, simplemente no se sabe.

Cuando se presenta el resultado 1 esto indica que el estimador de relevancia junto con el estimador de la cantidad de acciones a procesar tuvieron éxito en sus respectivas estimaciones. Si ocurre el resultado 2, llamado fracaso tipo I, indica lo contrario, alguno o ambos estimadores fallan en su predicción. Afortunadamente, está claro como recuperarse de un fracaso tipo I, simplemente procesando más acciones. Pero no está claro cuál es la alternativa más conveniente para recuperarse del resultado 3, llamado fracaso tipo II. Una opción es aumentar la cantidad de acciones procesadas con la esperanza de que esas nuevas acciones finalmente produzcan en la búsqueda un cambio tal que ahora esta sí finalice indicando si hay o no un plan. Esta alternativa de alguna manera supone un bias hacia la hipótesis de "no había un plan pues faltan acciones". Alternativamente, se puede optar por disminuir la cantidad de acciones a procesar con la esperanza de que la falla de la búsqueda se produjo debido a que la tarea parcialmente procesada ya era demasiado grande para ser trata por la búsqueda, por lo tanto, debemos achicarla. Esta alternativa tiene un bias hacia la hipótesis de "había un plan pero sobran acciones". Finalmente, hemos optado por recuperarnos de un fracaso tipo II de igual manera que de un fracaso tipo I, es decir, nos inclinamos a suponer que el origen del fracaso del proceso de grounding parcial se debe a que faltan acciones por procesar.

La sola posibilidad de que ocurra un fracaso tipo I, implica que la técnica de grounding parcial *no es completa*. Para lograr completitud, proponemos realizar un proceso de grounding incremental, donde si la tarea parcialmente procesada presenta un fracaso (de cualquier tipo) en la verificación, entonces incrementamos la cantidad de acciones a procesar y verificamos la tarea resultante, así sucesivamente hasta que se presente el resultado 1. Este método incremental, en el peor de los casos, termina cuando la cantidad de acciones a procesar es igual a RG^+. Llegado este caso el proceso de grounding incremental habrá convergido al mismo resultado que se obtiene mediante el proceso de grounding total y esto asegura trivialmente que las acciones procesadas contienen un plan. En definitiva, mediante grounding incremental logramos que la técnica sea completa en el límite.

[2] Obviamente suponiendo que la tarea totalmente procesada tenia al menos un plan.

7.2. Grounding Incremental

Una buena estimación de relevancia y cantidad de acciones a procesar no asegura completamente que las acciones resultantes contengan un plan de la tarea convirtiendo al algoritmo de grounding parcial en un algoritmo no completo. Para contrarrestar esta deficiencia proponemos ejecutar el algoritmo de grounding parcial en forma iterativa, donde en cada iteración, incrementamos la cantidad de acciones a procesar siempre que en la iteración anterior la tarea parcialmente procesada hasta ese momento no se pudo verificar que efectivamente contenía un plan de la tarea. En otras palabras, la idea de grounding incremental surge como método de recuperación a las posibles imprecisiones o errores en las estimaciones de relevancia y/o cantidad de acciones a procesar (fracaso tipo I) y al problema de la verificación (fracaso tipo II). Entonces, debemos decidir como incrementar la cantidad de acciones a procesar en la siguiente iteración teniendo en cuenta el siguiente trade-off: si el incremento es muy grande esto puede llevar a que la tarea resultante agote los recursos computacionales provocando la falla del proceso de grounding. En el otro extremo, si el incremento es muy pequeño, entonces es muy probable que la verificación de la existencia de un plan también falle tal como ocurría en la iteración anterior.

En esta sección presentamos un incremento constante y otro logarítmico para tratar con dicho trade-off. El incremento constante consiste en incrementar la cantidad de acciones a procesar en cada iteración por un número fijo, por ejemplo, 10.000 acciones. Así, el algoritmo de grounding incremental en su i-ésima iteración ejecuta grounding parcial con $SG + i * 10000$ acciones a procesar, donde SG es el estimador de grounding suficiente si optamos por la estrategia 2 en el problema de la parada (ver definición 45). La debilidad de un incremento constante es que el valor elegido puede ser elevado en ciertas tareas y muy pequeño en otras. Es decir, no es sensible al tamaño de la tarea totalmente procesada que se está intentando resolver. Supongamos que RG^+ está en el orden de los millones de acciones, entonces es evidente que un incremento de 10.000 acciones es demasiado pequeño para que la estrategia incremental termine con éxito. Ahora si RG^+ está en el orden de las miles de acciones, un incremento de 10.000 acciones parece demasiado grande lo que provocaría que la estrategia incremental tienda a converger a grounding total con mucha facilidad. En definitiva, necesitamos un mecanismo que determine un nivel de granularidad del incremento de acuerdo a la tarea que se está intentando resolver. Para ello, proponemos un incremento logarítmico sensible al tamaño de la tarea. Este incremento se caracteriza por realizar al principio leves incrementos y luego a medida que se presentan fracasos en la verificación, aumenta los incrementos en mayor proporción hasta converger

finalmente en RG^+, en el peor de los casos. Debido a este comportamiento creciente, podemos pensar que este modelo de incremento asume que el fracaso en la verificación de las primeras iteraciones es por un error relativamente bajo, por lo tanto, al principio hay que realizar leves incrementos. Pero si los fracasos persisten entonces no podemos confiar mas en el estimador de partida SG y debemos buscar un valor más alejado del mismo. Al principio buscamos en una vecindad de SG y en caso de fallar consistentemente, entonces nos alejamos de este estimador acercándonos a RG^+. La cantidad de acciones a procesar en la i-esima iteración por incremento logarítmico es:

$$SG + \frac{(RG^+ - SG)}{2^{(M-i)}} \; con \; M = 10.\log_{SG} RG^+$$

Por otra lado, es importante resaltar que el problema de la verificación toma aún más relevancia cuando implementamos grounding incremental, ya que, en cada iteración realizamos una verificación distinta. Por lo tanto, a cada verificación debemos asignarle una fracción del tiempo total disponible. Es decir, debemos decidir cuanto tiempo estamos dispuesto a esperar en cada una de las posibles verificaciones que podrían llevarse a cabo. Por ejemplo, si contamos con un timeout de 30 minutos para resolver una tarea, debemos repartir de alguna manera esos 30 minutos entre las diferentes posibles iteraciones, donde cada una de ellas ejecutará el algoritmo de grounding parcial y su respectiva verificación. La distribución del tiempo total disponible entre las distintas verificaciones individuales debe hacerse teniendo en cuenta el siguiente trade-off: a menor tiempo de verificación individual entonces mayor probabilidad de que se presenten fracasos tipo II. En consecuencia, grounding incremental tenderá a converger a grounding total con mayor facilidad y esto es claramente un efecto no deseado. Por el otro lado, mayor tiempo de verificación individual implica que sólo las primeras verificaciones individuales tendrán tiempo para llevarse a cabo, en consecuencia, la técnica incremental contaría con escasa capacidad de recuperación con respecto al error en la estimación de relevancia y/o cantidad de acciones a procesar, que era justamente el objetivo final de esta estrategia incremental. Observar que esta dificultad no ocurre para el caso de la memoria, ya que, esta es reutilizable. En consecuencia, cada verificación individual puede utilizar una cantidad de memoria determinada sin competir entre ellas por este recurso. De esta forma, a cada verificación le podemos asignar el total de la memoria disponible sin problemas.

7.3. Grounding Condicional

El concepto de relevancia introducido en la sección 7.1.1 se basa exclusivamente en las propiedades individuales de una acción, sin tener en cuenta ninguna otra información respecto de otras acciones, por ejemplo, las que ya han sido procesadas por el algoritmo de grounding. Hasta ahora, la relevancia de un acción es independiente de las otras acciones. Surge la idea de utilizar información sobre lo ocurrido hasta el momento para asignar relevancia a las acciones logrando de esta manera que nuestra función de relevancia sea sensible al contexto. Esto parece tener mucho sentido, ya que, la relevancia de una acción puede ser modificada por la ocurrencia previa de alguna otra acción. Por ejemplo, en el dominio *Depots* es muy posible que una vez que ocurre la acción "$load(crate_1)$", la acción "$unload(crate_1)$"se vuelve necesaria para alcanzar la meta. En otras palabras, las acciones que ocurren en el presente condicionan las acciones del futuro. Esta idea retrotrae a la noción de probabilidad condicional [Miller and Freund, 1973] debido a que estamos intentado determinar que tan relevante es una acción a dado que ocurrió la acción b. La acción condicionante b puede cambiar sustancialmente la probabilidad de ocurrencia de la acción a.

En definitiva, queremos introducir una función de relevancia sensible al contexto, detectando ciertos patrones o dependencias entre acciones. En particular, las dependencias que vamos a detectar son del tipo "si la acción a ocurre en algún plan (óptimo) de la tarea, entonces b también ocurre en el plan en algún momento posterior a la ocurrencia de a. Este patrón lo denotamos con $a \Rightarrow b$. La presencia de este tipo de patrón indica que la relevancia de b es mayor una vez que ocurre a. Por lo tanto, en el algoritmo de grounding si procesamos la acción a es muy conveniente procesar la acción b lo antes posible por sobre el resto de las acciones. Para implementar esta idea necesitamos crear reglas condicionales que capturen este tipo de dependencias entre acciones de una tarea. Nuevamente, estas reglas se infieren de un conjunto de tareas de entrenamiento para las cuales el planificador finaliza exitosamente, es decir, son tareas donde el grounding total termina y la búsqueda devuelve un plan óptimo. Las reglas condicionales son extraídas directamente desde los planes óptimos de forma muy simple: para toda acción a que ocurre en un plan óptimo de la tarea construimos la regla condicional $a \Rightarrow b$ para toda acción b que ocurre en el plan óptimo después de a. Los mismos motivos por los cuales las reglas relacionales introducidas en la sección 7.1.1 necesitaban ser a nivel de esquemas y no a nivel de acciones instancidas, son válidos también para el caso de las reglas condicionales. Estas no pueden hacer referencia a objetos concretos de una tarea. Por lo tanto, cada regla $a \Rightarrow b$ extraída de un plan óptimo infiere o generaliza la regla $a[X] \Rightarrow b[X']$, donde cada

objeto de a es sustituído por una variable y cada objeto de b es sustituído por la misma variable (ligada) con la cual ese objeto fue sustituído en a ó con una nueva variable (libre) si el objeto no ocurre en a. Por ejemplo, la regla $a(o_1, o_2, o_1) \Rightarrow b(o_1, o_3)$ infiere la regla $a[x, y, x] \Rightarrow b[x, z]$. Diremos que la variable x del esquema $b[x, z]$ es ligada, mientras que la variable z es libre.

Definición 46 (Reglas Duras y Blandas) *Sea $a[X] \Rightarrow b[X']$ una regla condicional, entonces diremos que la regla tiene n-grados de libertad si el esquema $b[X']$ tiene n-variables libres. Diremos que una regla es "dura"si tiene 0-grados de libertad, en caso contrario, diremos que es una regla "blanda".*

Podemos utilizar las reglas condicionales en el algoritmo de grounding parcial de la siguiente manera: supongamos que contamos con una regla dura $a[x, y, z] \Rightarrow b[z, x]$ y acabamos de procesar la acción $a(o_1, o_2, o_3)$ que es una instanciación de la parte izquierda de la regla. Por la semántica de la regla condicional debemos procesar inmediatamente la acción $b(o_3, o_1)$. En conclusión las reglas duras determinan unívocamente la siguiente acción a procesar. Esta unicidad no sucede cuando se trata de reglas blandas producto de la ocurrencia de variables libres en la parte derecha de la regla. Para ilustrar esto, consideremos nuevamente la regla blanda $a[x, y, x] \Rightarrow b[x, z]$ y supongamos que acabamos de procesar la acción $a(o_1, o_2, o_1)$ que es una instanciación de la parte izquierda de la regla. No resulta obvio determinar cuál es la acción instanciada del esquema $b[x, z]$ que debemos procesar a continuación, ya que, la variable libre z puede ser instanciada a cualquier objeto. Surge la pregunta sobre cómo instanciar las variables libres en la reglas blandas. Una primera opción es instanciar con todos los objetos posibles pero esta estrategia tiene la clara desventaja de que a mayor grado de libertad de la regla blanda, más acciones son procesadas muchas de la cuales pueden ser irrelevantes para la tarea, y esto es contrario al objetivo final de la técnica, donde buscamos procesar la menor cantidad de acciones posibles que me lleven a un plan de la tarea. Otra opción es instanciar las variables libres en forma aleatoria con la desventaja de convertir el algoritmo de grounding parcial en "no-determinístico". Este es un tópico que actualmente está en desarrollo y esperamos encontrar prontamente mejores opciones para instanciar las variables libres que ocurren en las reglas blandas de un dominio.

Otro aspecto importante a considerar en relación a las reglas condicionales duras y/o blandas es su nivel de confiabilidad, una medida de qué tan "vinculante" una regla es para el proceso de grounding. Una regla será más confiable a medida que más veces esta se verifica en el conjunto de tareas de entrenamiento. Para esto, introducimos la noción de instancias positivas y negativas de una regla.

Definición 47 (Instancias Positivas y Negativas) *Sean a,b acciones instanciadas y $a[X] \Rightarrow b[X'])$ una regla condicional inferida del conjunto de tareas de entrenamiento. Dado un plan óptimo, diremos que $a \Rightarrow b$ es una instancia positiva de la regla, si a y b son instanciadas como indican las variables de la regla y b ocurre en el plan óptimo luego de la ocurrencia de a. Diremos $a \Rightarrow b$ es una instancia negativa de la regla, si a y b son instanciadas como indican las variables de la regla y b no ocurre en el plan óptimo luego de la ocurrencia de a.*

Un regla es más confiable mientras mayor es la proporción de instancias positivas sobre la cantidad total de instancias. Esta proporción, llamado *coeficiente de confiabilidad*, es utilizado para filtrar aquellas reglas que tienen un coeficiente de confiabilidad inferior a una cierta constante $C \in [0, 1]$. Las reglas con mejor confiabilidad son las únicas consideradas en el proceso de grounding. En el ejemplo 4, se muestra una de las reglas condicionales extraídas del conjunto de tareas de entrenamiento del dominio *Satellite*. Se trata de un regla dura, ya que, la parte derecha de la regla no tiene variables libres. Además, esta regla tiene 2 instancias positivas y 4 negativas, derivando esto en un coeficiente de confiabilidad de 0.33. Lo cual, representa un valor bastante elevado dados los valores observados para otras reglas.

Ejemplo 4 *Regla condicional dura del dominio* Satellite *inferida del conjunto de tareas de entrenamiento con coeficiente de confiabilidad de 0.33 producto de dos instancias positivas (+) y cuatro negativas (-).*

```
0.33 take_image[x,y,z,w] => switch_off[z,x]

(+) take_image(satellite0,phenomenon5,instrument1,infrared1)
    => switch_off(instrument1,satellite0)

(+) take_image(satellite1,phenomenon1,instrument1,infrared1)
    => switch_off(instrument1,satellite1)

(-) take_image(satellite0,phenomenon6,instrument0,thermograph3)
    => switch_off(instrument0,satellite0)

(-) take_image(satellite0,planet2,instrument0,thermograph0)
    => switch_off(instrument0,satellite0)

(-) take_image(satellite0,star3,instrument0,thermograph2)
    => switch_off(instrument0,satellite0)
```

```
(-) take_image(satellite0,phenomenon0,instrument0,thermograph3)
    => switch_off(instrument0,satellite0)
```

En el caso particular de esta regla, si consideramos que un coeficiente de 0.33 es suficiente para que la misma sea tenida en cuenta en el proceso de grounding, al procesar una acción instanciada que coincide con la parte izquierda de la regla, por ejemplo, $take\_image(o_1, o_2, o_3, o_4)$, inmediatamente instanciamos las variables ligadas de la parte derecha de la misma con los objetos correspondientes, obteniendo en este caso, la acción $switch\_off(o_3, o_1)$ para ser procesada lo más pronto posible por el algoritmo. Pero la acción $switch\_off(o_3, o_1)$ sólo puede ser procesada a partir del momento en que sus precondiciones están definidas en F, no antes. En caso contrario, las reglas condicionales llevarían al algoritmo de grounding a no satisfacer la definición 43 de *fact-consistencia*. Lo que podría ocasionar, como ya vimos, la existencia de acciones cuyas precondiciones no están definidas. En otras palabras, debemos esperar a que la acción correspondiente a la parte derecha de la regla sea relajadamente alcanzable. Para implementar esto, tenemos una segunda cola donde encolamos las acciones pendientes de procesar que fueron instanciadas por derecha producto de las reglas condicionales. Cuando sus precondiciones son procesadas, entonces estas acciones pendientes son inmediatamente procesadas por el algoritmo.

7.4. Trabajos Relacionados.

En la literatura de planning podemos encontrar diferentes técnicas previas a nuestra técnica de subdominización que intentan aliviar el proceso de grounding de un planificador. Algunas se basan en evitar procesar facts y acciones no alcanzables desde el estado inicial [Helmert, 2009]. También, algunas utilizan la noción de simetrías para evitar trabajo redundante durante el proceso de grounding [Röger *et al.*, 2018]. O inclusive nuestra propia técnica Action Schema Splitting la cual reformula la descripción PDDL del dominio dividiendo los esquemas de acción (con múltiples párametros) en sub-esquemas [Areces *et al.*, 2014]. Aunque existen técnicas que evitan completamente el proceso de grounding, y por ende, realizan la búsqueda heurística directamente sobre la representación esquemática de la tarea, es decir, a nivel de esquemas de acción [Penberthy and Weld, 1992]. Estas han perdido popularidad debido a que la representación proposicional de una tarea hace más fácil el cómputo de heurísticas informadas. Por ejemplo en [Ridder and Fox, 2014] adaptan la heurística delete-relaxation [Hoffmann and

Nebel, 2001a] a nivel esquemático. Existen varias alternativas que intentan eliminar facts y acciones irrelevantes pero a partir de la tarea proposicional completamente procesada. [Nebel *et al.*, 1997; Hoffmann and Nebel, 2001b; Haslum *et al.*, 2013; Torralba and Kissmann, 2015]. Tal vez, lo más cercano a subdominización es la técnica *under-approximation refinement* [Heusner *et al.*, 2014], la cual, también realiza una búsqueda con sólo un subconjunto de acciones. Sin embargo, todas estas técnicas utilizan información de la tarea totalmente procesada para después determinar el subconjunto de acciones con la cuales se lleva a cabo finalmente la búsqueda. Por lo tanto, estas técnicas no son aplicables en aquellas tareas donde un proceso de grounding total es inviable, a diferencia de nuestra técnica que permite tratar con este tipo de tareas.

Los resultados empíricos en el capítulo 8 demuestran que aplicar métodos de ML para identificar las acciones relevantes de una tarea es un tópico muy prometedor dentro de la comunidad de planning. De hecho, recientemente, [Toyer *et al.*, 2018] introduce una técnica basado en ML para obtener funciones heurísticas para el proceso de búsqueda del planificador para dominios específicos. La diferencia con nuestro trabajo radica en que la heurística aprendida brinda información sobre los estados relevantes de una tarea y no de las acciones propiamente. Más aún, los autores utilizaron redes neuronales en lugar de nuestros clásicos modelos de ML.

Capítulo 8

Evaluación: experimentos y resultados

Para la evaluación de nuestra técnica de grounding parcial adaptamos la implementación de la componente *"translator"* del planificador Fast Downward (FD) [Helmert, 2006]. Esta componente parsea el archivo PDDL dado como entrada y devuelve la tarea de planning obtenida mediante grounding total en una representación FDR [Bäckström and Nebel, 1995; Helmert, 2009]. Nuestras modificaciones son mínimamente invasivas porque sólo cambiamos el orden en que las acciones son procesadas y la condición de terminación del algoritmo en las diferentes formas ya mencionadas en la secciones 7.1.1 y 7.1.2, respectivamente. Por lo tanto, ninguna de estas modificaciones implementadas afecta la correctitud de la componente *translator* de FD, es decir, la tarea obtenida por grounding parcial siempre es una tarea FDR propiamente dicha. Además, los cambios implementados no afectan la velocidad de la componente, por supuesto, siempre que no utilicemos un modelo de relevancia muy costoso al momento de computar su estimación.

8.1. Acerca del benchmarks

Para los experimentos requerimos dominios para los cuales se encuentran disponibles sus respectivos generadores de tareas y además su tamaño crezca considerablemente con respecto a los parámetros del generador. Esto con el objetivo de generar para cada dominio dos conjuntos de tareas distintos. Un conjunto con tareas cuyo tamaño permite que el planificador finalice exitosamente hallando un plan de la tarea, y por consiguiente, estas constituyen el conjunto de tareas de entrenamiento. Por otro lado, un conjunto de tareas de mayor tamaño para las cuales el proceso de grounding total falla, y de

esta manera, construir un conjunto de tareas de evaluación interesantes para observar la efectividad de la técnica.

Concretamente, escogimos cuatro dominios que fueron parte del "learning track" de la competencia internacional de planning (IPC) del año 2011: *Blocksworld*, *Depots*, *Satellite* y *TPP*. Así como también dos dominios del "deterministic track" IPC 18: *Agricola* y *Caldera*. Para los seis dominios elegidos, utilizamos las tareas que fueron generadas para el track determinístico IPC, y además, un conjunto de 25 tareas más grandes, denominadas "large", que fueron generadas por nosotros mismos para los experimentos. Para el entrenamiento de los modelos, empleamos entre 40 y 250 tareas con el fin de obtener suficientes datos de entrenamiento para cada uno de los esquemas de acción. Cabe aclarar que como consecuencia de que el número de acciones perteneciente a un esquema de acción varía significativamente a través de los distintos dominios, entonces fijamos un número distinto de tareas de entrenamiento para cada dominio en particular.

Para generar tareas large comenzamos con el tamaño de la tarea IPC más grande, escalando los parámetros del generador linealmente. Por ejemplo, en el dominio *Satellite* la tarea IPC más grande posee alrededor de 10 satélites y 20 instrumentos. Esto a su vez, es el tamaño más chico de las tareas generadas por nosotros, llegando a generar tareas con hasta 15 satélites y 60 instrumentos. En *Blocksworld*, donde las tareas IPC escalan hasta 17 bloques, nosotros decidimos escalar comenzando desde 75 bloques hasta 100, ya que, a pesar de ser una cantidad de bloques bastante importante nuestra técnica aún logra resolverlos con relativa facilidad. Con respecto a los dominios, utilizamos la codificación tipada de *Satellite* correspondiente al learning track. En *Blocksworld*, empleamos la versión "no-arm", el cual presenta una explosión cúbica en el tamaño de la tarea, en contraste, con la versión "arm", donde el tamaño de la tarea es cuadrática en la descripción PDDL.

8.2. Modelos ML

Experimentamos primero con funciones de relevancia desinformada, tales como, FIFO, LIFO, Random para intentar dar evidencia hasta donde es posible llegar sin la implementación de técnicas de ML. Luego, experimentamos con funciones de relevancia informadas basadas en aprendizaje utilizando modelos de clasificación-regresión, con el objetivo de detectar aquellas acciones que resultan ser más importantes para resolver la tarea. En todos los caso, combinamos estas funciones de relevancia con una política Round Robin (RR). Reportamos resultados para los modelos *logistic regression classifier* (**LOGR**), *kernel ridge regression* (**KRN**), *linear regression* (**LINR**) y *sup-*

port vector machine regressor (**SVR**). Consideramos LINR y LOGR por su simplicidad, son modelos lineales que combinan los diferentes atributos de un vector. Por su parte, KRN y SVR fueron considerados por ser modelos no-lineales que suelen ser útiles en aquellos casos donde modelos lineales son poco precisos en su predicción. Para esto utilizamos las implementaciones ya existentes de estos modelos en el paquete Python `scikit` [Pedregosa *et al.*, 2011], el cual, es fácilmente adaptable al código fuente de la componente translator del planificador FD. Para la selección de atributos donde seleccionamos aquellas reglas que son más útiles para estimar la probabilidad de que una acción pertenezca a un plan óptimo de la tarea, utilizamos la información adicional que devuelven los modelos entrenados. Por ejemplo, para cada atributo (o sea, regla relacional), el modelo entrenado retorna un valor numérico que especifica la capacidad del atributo de discriminar la clase (target). Este valor se utiliza para filtrar los atributos que finalmente van a constituir el vector de atributos final.

La Figura 8.1 muestra los resultados de la función de relevancia obtenida por un modelo LOGR para el dominio *Blocksworld* sobre un conjunto de tareas de evaluación, especialmente generadas de forma tal que su tamaño permita al planificador obtener un plan óptimo. El eje horizontal representa el intervalo [0,1] divido en sub-intervalos y el eje vertical representa cantidad de acciones. Cada barra indica la cantidad de acciones que tuvieron asignada una relevancia estimada en el subintervalo indicado, resaltando en color azul aquellas que pertenecen a un plan óptimo de la tarea. La primera observación sobre esta gráfica indica que la función de relevancia LOGR funciona adecuadamente para las acciones del esquema *move-t-to-b* y *move-b-to-t*, donde la función pudo discriminar las acciones optimales, es decir, aquellas que pertenecen a un plan óptimo de la tarea, de las que no lo son (acciones no-optimales). Esta resultado no ocurre, al menos con tanta claridad, para las acciones correspondientes al esquema *move-b-to-b*. Otra observación es que este último esquema de acción cuenta con una mayor proporción de acciones generadas con respecto a los otros dos esquemas del dominio. Al proyectar estas observaciones, esperamos que el modelo LOGR para este dominio priorizará las acciones provenientes de los esquemas *move-t-to-b* y *move-b-to-t* por sobre las acciones del esquema *move-b-to-b*. Los primeros dos esquemas producen las acciones necesarias para resolver cualquier tarea de *Blocksworld*. Mientras que el último esquema mencionado produce las acciones necesarias para resolver una tarea en forma óptima. Otra vez, observando la gráfica, si procesamos solamente las acciones con una relevancia estimada mayor a 0.6, lo cual, implica descartar todas las acciones del esquema *move-b-to-b* y muchas de las acciones no-optimales de los otros dos esquemas, esto parece ser suficiente para resolver la tarea. Por supuesto, cuando trabajamos con

técnicas de ML, siempre tenemos el riego de *overfitting*. En este caso los resultados en las tareas de entrenamiento son muy similares a los resultados de las tareas de validación mostrado en la Figura 8.1. Esto sugiere que el problema de overfitting no parece ser una complicación presente en nuestro setup, y más aún, teniendo en cuenta que los resultados en otros dominios fueron similares a *Blocksworld*.

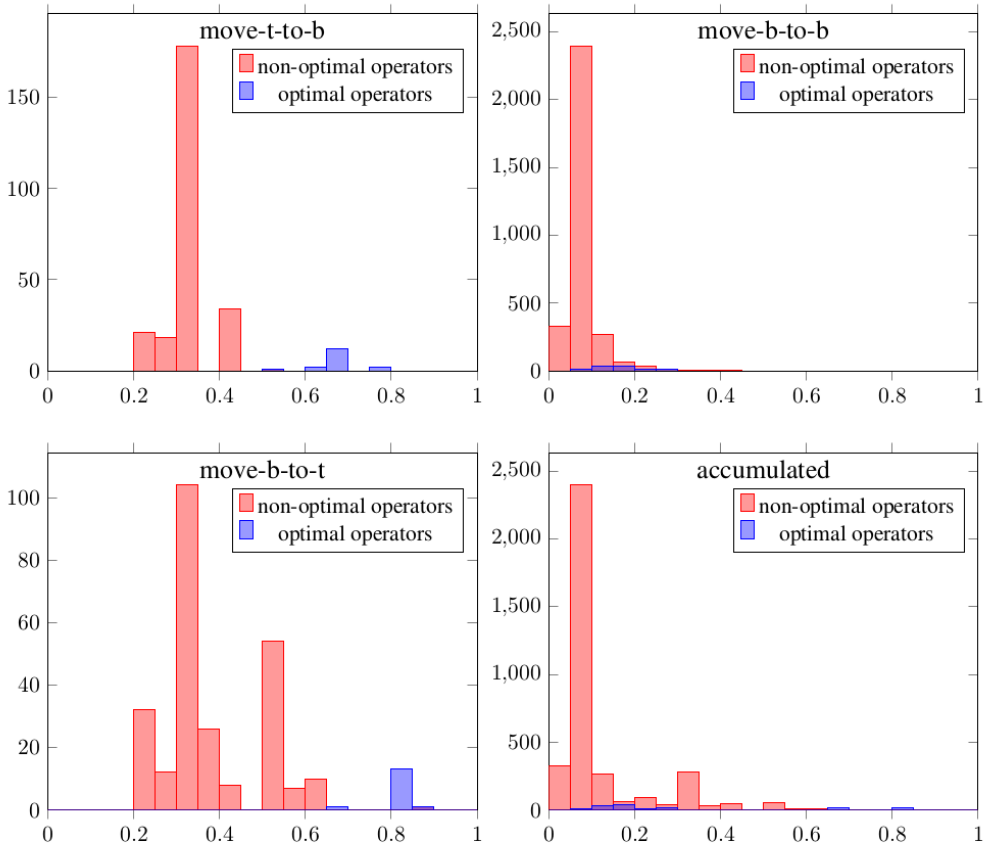


Figura 8.1: *Evaluación del modelo LOGR en el dominio* Blocksworld. *En el eje vertical representamos los valores devueltos por la función de relevancia estimada basada en el modelo LOGR y en el eje horizontal la cantidad de acciones que resultaron tener la relevancia indicada en el intervalo correspondiente. Las acciones optimales (aquellas que pertenecen a un plan óptimo) se resaltan en azul en contraste con las acciones no optimales en color rojo.*

8.3. Baseline vs Grounding Incremental

El baseline de nuestros experimentos consiste en la ejecución del algoritmo de grounding total y una búsqueda heurística sobre la tarea totalmente procesada conocida como "LAMA First Iteration" [Richter and Westphal, 2010], por ser una buena configuración estándar para *satisfacing planning* y por ser fácilmente utilizable al encontrarse bien integrada en el planificador FD. Todo este proceso de grounding y búsqueda restringido a un timeout de 30 minutos y 4GB de memoria. Con respecto a grounding incremental, la primer iteración procesamos acciones hasta que la meta es alcanzada relajadamente, es decir, procesamos las GG^+ acciones más relevantes y luego realizamos su verificación. Si la verificación pertinente falla, entonces incrementamos el número de acciones a procesar en 10.000 acciones, así sucesivamente hasta hallar un grounding parcial exitoso. Permitimos un total de 5 horas y 4GB para la técnica de grounding incremental donde restringimos cada verificación a 10 minutos para mantener manejables los tiempos de ejecución de los experimentos. Esto claramente no logra asegurar el cubrimiento del rango $[GG^+, RG^+]$ pero si es útil para hallar con cierta precisión, el mínimo número de acciones que se necesitan procesar para resolver una tarea. Por supuesto, esto con un nivel de granularidad fijado arbitrariamente en 10.000 acciones.

En planning el valor coverage de un dominio representa la cantidad de tareas resueltas bajo un cierto límite de tiempo y memoria. En el cuadro 8.1 mostramos los valores de *"coverage"* obtenidos para los seis dominios del benchmark mencionado. En el baseline (**Base**) una tarea es resuelta si el planificador encontró un plan en a lo sumo 30 minutos y 4GB de memoria. En el caso de grounding incremental consideramos una tarea como resuelta si el planificador encontró un plan dentro de los primeros 30 minutos del proceso incremental entero (5 horas). Se observa que para las tareas IPC, el baseline resuelve casi la totalidad de las mismas, salvo en el dominio *Agricola* y *Caldera*. El baseline tiene serias dificultades para tareas "large" debido, en la mayoría de los casos, a una falla en el proceso de grounding total. Quizás lo más importante del cuadro es notar que siempre existe una función de relevancia obtenida mediante ML (LINR, LOGR, KRN o SVR) que incrementa el coverage con respecto al baseline para cada dominio excepto *TPP*. Por ejemplo, para el dominio *Blocksworld* sorprendentemente pasamos de un coverage de 0 a 25 mediante LINR-RR, LOGR, KRN y SVR. Para el dominio *Caldera* aumentamos el coverage de 0 a 23 por LOGR-RR. Para el dominio *Agricola* el aumento fue de 4 a 22 por SVR-RR. Para el dominio *Satellite* pasamos de un coverage de 0 a 15 por KRN. Estos dominios dan evidencia del poder de nuestra técnica cuando el modelo aprendido captura los atributos

más importantes de aquellas acciones que son más relevantes para resolver una tarea. Si consideramos el total de la tareas, SVR-RR obtuvo el valor más alto de coverage con 233 tareas resueltas, lo que representa una mejora importante con respecto a las 156 por parte del baseline. También se observa que las funciones de relevancia no informadas (LIFO, FIFO, RND) parecen no funcionar mejor que el baseline, salvo el caso puntual del dominio *TPP*, donde tenemos un mínimo aumento del coverage de 7 a 8 con LIFO.

			Tareas resueltas en los primeros 30 minutos											
						LINR		LOGR		KRN		SVR	
Dominios	#	Base	FIFO	LIFO	RND		RR		RR		RR		RR
Agricola-IPC	20	**10**	1	1	2	1	7	5	5	8	5	4	**10**
Agricola-large	25	4	0	0	0	0	1	2	1	0	1	0	**22**
Blocksworld-IPC	35	35	35	35	35	35	35	35	35	35	35	35	35
Blocksworld-large	25	0	0	0	0	14	**25**	**25**	23	**25**	24	**25**	22
Caldera-IPC	20	13	13	12	13	17	18	18	18	11	18	**19**	18
Caldera-large	25	0	10	0	3	22	18	18	**23**	1	19	20	17
Depots-IPC	22	20	19	20	19	20	20	19	**21**	19	20	20	**21**
Depots-large	25	1	0	0	0	**5**	3	1	2	2	3	1	4
Satellite-IPC	36	**36**	35	**36**	**36**	**36**	35	**36**	35	35	35	**36**	35
Satellite-large	25	0	0	0	0	0	11	0	14	**15**	14	0	14
Tpp-IPC	30	**30**	**30**	**30**	**30**	26	28	**30**	**30**	**30**	**30**	**30**	29
Tpp-large	25	7	5	**8**	2	1	2	4	4	5	6	4	6
$\sum$	313	156	148	142	140	177	203	193	211	186	210	194	**233**

Cuadro 8.1: *Coverage considerando una tarea como resuelta si lo fue en los primeros 30 minutos del proceso de grounding incremental entero (5 horas). Coverage obtenidos para baseline (Base); grounding incremental con función de relevancia no informada (FIFO, LIFO, random (RND)); grounding incremental con función de relevancia informada basadas en ML (LINR, LOGR, KRN, SVR). "RR" indica round robin, el cual, fue implementado mediante una cola distinta para cada esquema de acción. "#" cantidad de tareas. Mejor valor de coverage resaltado en negrita.*

Si focalizamos los resultados del cuadro 8.1 exclusivamente sobre tareas "large"se puede apreciar aún con mayor claridad que la técnica de grounding incremental tiene mejoras muy significativas con respecto al baseline. En el cuadro 8.2 tomamos el porcentaje de coverage, es decir, el coverage sobre la cantidad de tareas evaluadas del dominio. Este porcentaje se muestra para cada heurística de la tabla 8.1 anterior. Por ejemplo, para el dominio *Agricola* tenemos un 16 % de las tareas large resueltas a través del baseline y aumentamos a un 88 % mediante SVR-RR. En *Blocksworld* tenemos un aumento de este porcentaje del 0 % al 100 % con LINR-RR, LOGR, KRN,

y SVR. En *Caldera*, tenemos una mejora de 0 % a 92 % mediante LOGR-RR. En *Satellite* la mejora es del 0 % al 60 % por KRN. En *Depots* si bien la mejora es menor con respecto a la ocurrida en los dominios anteriores, aún es una mejora considerable, del 4 % al 20 % por LINR. El dominio *TPP*, registra la menor mejora, del 28 % al 32 % y curiosamente producto de una heurística no informada (LIFO). Por último, sobre todas los tareas large, el estimador de relevancia SVR-RR registra el mejor porcentaje de coverage con un 57 %, lo cual, representa una gran mejora con respecto al escaso 8 % por parte del baseline.

			% Coverage en los primeros 30 minutos									
						LINR		LOGR		KRN		SVR
Dominios	#	Base(%)	FIFO	LIFO	RND		RR		RR		RR		RR
Agricola	25	16	0	0	0	0	4	8	4	0	4	0	**88**
Blocksworld	25	0	0	0	0	56	**100**	**100**	92	**100**	96	**100**	88
Caldera	25	0	40	0	12	88	72	72	**92**	4	76	80	68
Depots	25	4	0	0	0	**20**	12	4	8	8	12	4	16
Satellite	25	0	0	0	0	0	44	0	56	**60**	56	0	56
TPP	25	28	20	**32**	8	4	8	16	16	20	24	16	24
Total	150	8	10	5	3	28	40	33	45	32	45	33	**57**

Cuadro 8.2: *Porcentaje de Coverage obtenidos sobre tareas "large"para baseline (Base); grounding incremental con función de relevancia no informada (FIFO, LIFO, random (RND)); grounding incremental con función de relevancia informada basadas en ML (LINR, LOGR, KRN, SVR). "RR" indica round robin el cual fue implementado mediante una cola distinta para cada esquema de acción. "#" cantidad de tareas. Mejor porcentaje de coverage resaltado en negrita.*

En el cuadro 8.3 mostramos los valores *"coverage"* para los mismos seis dominios del cuadro 8.1, pero tratando de analizar que sucedería con estos valores si tuviésemos una condición de parada perfecta, es decir, una estimación precisa de la cantidad de acciones a procesar necesarias para que la tarea parcialmente procesada contenga un plan de la tarea. Para ello, vamos a considerar una tarea como resuelta si la última iteración del proceso de grounding incremental fue exitosa (el planificador halló un plan) en a lo sumo 30 minutos. De esta manera, los valores de coverage de este cuadro deben ser mejores, es decir, mayores o iguales, a los observados en el cuadro 8.1. Para el dominio *Caldera* donde teníamos un coverage de 22 para tareas large con LOGR-RR, ahora alcanzamos un 25. Para el dominio *Agricola* tenemos un incremento de 22 a 24 tareas resueltas con SVR-RR. Para el dominio *Satellite* tenemos un incremento de 15 a 19. Para el dominio *TPP* aumentamos levemente el coverage de 8 obtenido por LIFO a 9 pero ahora obtenido

por SVR-RR. Otra vez, SVR-RR tiene el mejor comportamiento general, y aumentamos su coverage de 233 a 255 tareas. Esta última tabla sugiere la importancia de contar con una buena condición de parada y esto es lo que se busca mediante el estimador *SG* (ver sección 7.1.2).

			Tareas resueltas en la última iteración en 30 minutos											
						LINR		LOGR		KRN		SVR	
Dominios	#	Base	FIFO	LIFO	RND		RR		RR		RR		RR
Agricola-IPC	20	**10**	9	9	9	9	**12**	11	**12**	9	**12**	10	**12**
Agricola-large	25	4	4	4	4	1	**24**	20	23	6	**24**	19	**24**
Blocksworld-IPC	35	35	35	35	35	35	35	35	35	35	35	35	35
Blocksworld-large	25	0	0	0	0	14	**25**	**25**	24	**25**	**25**	**25**	24
Caldera-IPC	20	13	17	13	17	19	19	19	19	17	19	**20**	19
Caldera-large	25	0	19	0	5	**25**	**25**	24	**25**	8	**25**	**25**	**25**
Depots-IPC	22	20	19	20	19	20	20	19	**21**	19	20	20	**21**
Depots-large	25	1	0	0	0	**5**	3	1	2	2	3	1	4
Satellite-IPC	36	**36**	35	**36**	**36**	**36**	**36**	**36**	**36**	**36**	**36**	**36**	**36**
Satellite-large	25	0	0	0	0	0	14	0	16	**19**	15	0	16
Tpp-IPC	30	**30**	30	30	30	30	30	30	30	30	30	30	30
Tpp-large	25	7	6	8	6	5	5	7	5	5	6	7	**9**
$\sum$	313	156	174	155	161	199	248	227	248	211	250	228	**255**

Cuadro 8.3: *Coverage obtenidos considerando la última iteración resuelta en 30 minutos para baseline (Base); grounding incremental con función de relevancia no informada (FIFO, LIFO, random (RND)); grounding incremental con función de relevancia informada basadas en ML (LINR, LOGR, KRN, SVR). "RR" indica round robin el cual fue implementado mediante una cola distinta para cada esquema de acción. "#" cantidad de tareas. Mejor valor de coverage resaltado en negrita.*

Nuevamente focalizamos los resultados del cuadro 8.3 exclusivamente sobre tareas "large" para apreciar en términos porcentuales la mejora obtenida por la técnica de grounding incremental si consideramos la última iteración del algoritmo. Entonces, en el cuadro 8.4 presentamos el porcentaje de coverage de la última iteración del algoritmo. Por ejemplo, para el dominio *Agricola* aumentamos el porcentaje de coverage de 88 % al 96 % por KRN-RR y SVR-RR. En *Blocksworld* teníamos un 100 % y ahora, obviamente también pero con 5 diferentes heurísticas, mientras que antes lo hacíamos con 4 heurísticas distintas. En *Caldera* aumentamos del 92 % al 100 % y con 6 heurísticas distintas. En *Satellite* mejoramos de un 60 % a un 76 % en KRN. En *TPP* aumentamos levemente de 32 % a 36 %, pero ahora mediante SVR-RR. En *Depots* no logramos aumentar el porcentaje de coverage, se mantiene en 20 %

			% Coverage última iteración en 30 minutos											
						LINR		LOGR		KRN		SVR	
Dominios	#	Base(%)	FIFO	LIFO	RND		RR		RR		RR		RR
Agricola	25	16	16	16	16	4	96	80	92	24	**96**	76	**96**
Blocksworld	25	0	0	0	0	56	**100**	**100**	96	**100**	**100**	**100**	96
Caldera	25	0	76	0	20	**100**	**100**	96	**100**	32	**100**	**100**	**100**
Depots	25	4	0	0	0	**20**	12	4	8	8	12	4	16
Satellite	25	0	0	0	0	0	56	0	64	**76**	60	0	64
TPP	25	28	24	32	24	20	20	28	20	20	24	28	**36**
$\sum$	150	8	19	8	10	33	64	51	63	43	65	51	**68**

Cuadro 8.4: *Porcentaje de Coverage obtenidos sobre tareas "large" considerando la última iteración resuelta en 30 minutos para baseline (Base); grounding incremental con función de relevancia no informada (FIFO, LIFO, random (RND)); grounding incremental con función de relevancia informada basadas en ML (LINR, LOGR, KRN, SVR). "RR" indica round robin el cual fue implementado mediante una cola distinta para cada esquema de acción. "#" cantidad de tareas. Mejor porcentaje de coverage resaltado en negrita.*

con LINR. Considerando todos las tareas, el estimador de relevancia SVR-RR era el que registraba el mayor porcentaje de coverage con un 57 % y esto fue mejorado a un 68 %. Estos porcentajes terminan de evidenciar en forma contundente la importancia de contar con un buen estimador de acciones suficientes a procesar para hallar un plan de la tarea. En otras palabras, resulta clave tener una condición de parada del algoritmo incremental lo más precisa posible. En la sección 8.4 evaluamos el estimador *SG* definido en la ecuación 7.2 de la sección 7.1.2. Lo cual es un primer intento por conseguir un estimador de calidad que indique la cantidad de acciones que debe procesar el algoritmo para hallar un plan.

Luego de analizar el impacto que tuvo la implementación de grounding incremental sobre el coverage en el benchmark, ahora nos concentraremos en estudiar el impacto en la cantidad de acciones procesadas que fueron necesarias para hallar un plan de la tarea. Es decir, queremos ver si la técnica incremental consigue obtener una reducción considerable de las acciones procesadas con respecto a grounding total. Para esto, los ploteos de las figuras 8.2, 8.3, 8.4, 8.5, 8.6, 8.7 muestran para cada dominio del benchmark el número de acciones que fueron procesadas hasta hallar un plan de la tarea mediante grounding incremental con diferentes funciones de relevancia. Cada punto se corresponde con una tarea del dominio. El eje vertical representa la cantidad de acciones procesadas por grounding incremental en su última iteración si fue exitosa y el eje horizontal representa la cantidad de accio-

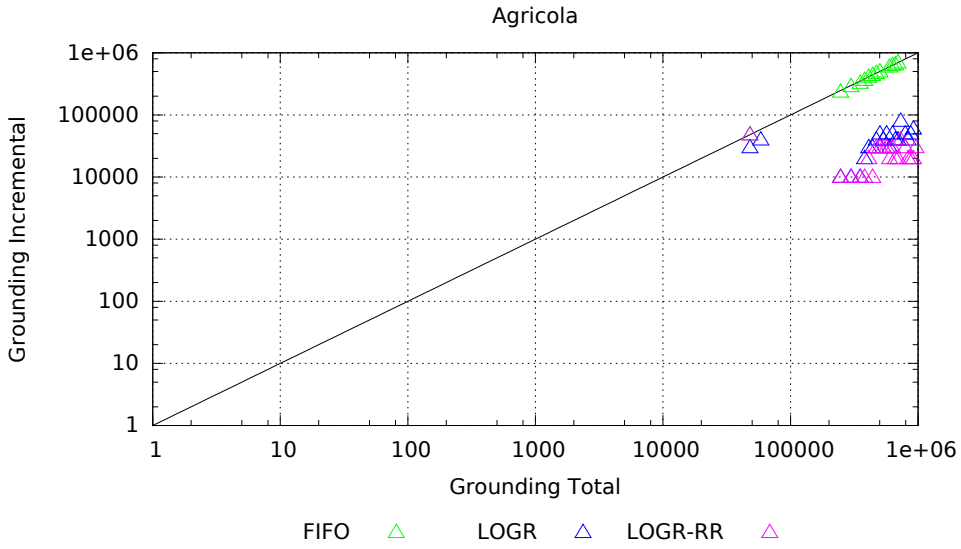


Figura 8.2: Cantidad de acciones procesadas por grounding incremental vs grounding total en el dominio *Agricola*.

nes procesadas por grounding total (RG^+). Si la función de relevancia logra capturar aquellas acciones verdaderamente relevantes de la tarea, entonces esperamos ver una gran concentración de puntos en la parte inferior derecha de estos ploteos, ya que, por la definición de los ejes, eso indica que el proceso de grounding incremental logró hallar un plan de la tarea procesando una menor cantidad de acciones con respecto a grounding total. Los puntos sobre la diagonal de la gráfica se corresponden con aquellas tareas en donde no hubo una reducción en la cantidad de acciones procesadas por grounding incremental. Por cuestiones de fácil visualización decidimos no graficar todas las curvas correspondientes a las funciones de relevancia exhibidas en las tablas 8.1 y 8.3 sino solamente la mejor de ellas, es decir, aquella función de relevancia que consigue una mayor reducción en el número de acciones procesadas. Más precisamente, para cada dominio en particular graficamos la curva correspondiente a la mejor función de relevancia entre las no informadas (LIFO, FIFO, RND); la curva correspondiente a la mejor función de relevancia entre las informadas (LINR, LOGR, KRN,SVR) y su respectiva versión round robin (RR). Por ejemplo, para el dominio *Agricola* la mejor función de relevancia no informada fue FIFO y la mejor función de relevancia informada fue LOGR. Por lo tanto, en la gráfica correspondiente al dominio *Agricola* únicamente graficamos las curvas de FIFO, LOGR y LOG-RR.

En la figura 8.2 correspondiente al dominio *Agricola* se observa que FIFO

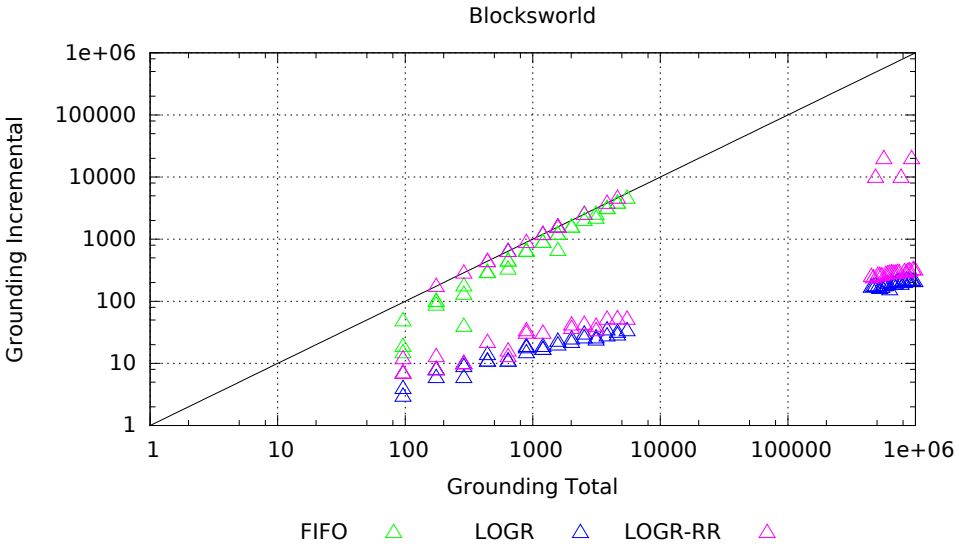


Figura 8.3: Cantidad de acciones procesadas por grounding incremental vs grounding total en el dominio *Blocksworld*.

no reduce la cantidad de acciones, la mayoría de sus puntos se encuentran sobre la diagonal. Sin embargo, si se observa una reducción para el caso de la estimadores de relevancia LOGR y LOGR-RR. Estos estimadores redujeron el orden de la acciones por debajo de las 100 mil cuando se encontraban cerca del millón. La opción LOGR con round robin (LOGR-RR) parece funcionar un poco mejor que su versión sin round robin (LOGR).

En la figura 8.3 correspondiente al dominio *Blocksworld* se observa que FIFO no reduce la cantidad de acciones, salvo algunas pocas tareas. Si se observa una reducción muy importante para el caso de la estimadores de relevancia LOGR y LOGR-RR. Por una lado, para las tareas que tenían una cantidad de acciones que iban entre 100 y 10 mil, se redujeron al rango entre 10 a 100 acciones. Por otro lado y el más importante de destacar es el de las tareas que estaban en un orden cercano al millón de acciones, que ahora se encuentran por debajo de las mil acciones (excepto en 4 tareas). Claramente, en este dominio podemos decir que la técnica incremental fue un éxito.

En la figura 8.4 correspondiente al dominio *Caldera* se observa una reducción significativa de la cantidad de acciones con FIFO, aunque esta reducción es menor en comparación con las obtenidas por LOGR y LOGR-RR que lograron una reducción entre 1 a 2 órdenes de magnitud. Nuevamente, podemos decir que en este dominio las funciones de relevancia LOGR y LOGR-RR funcionaron, al igual que en el caso *Blocksworld*, exitosamente.

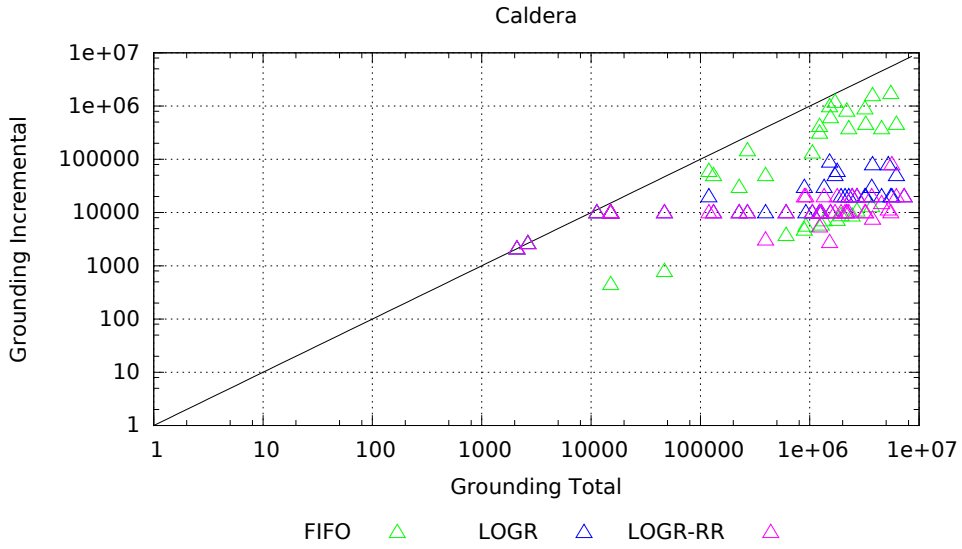


Figura 8.4: Cantidad de acciones procesadas por grounding incremental vs grounding total en el dominio *Caldera*.

En la figura 8.5 correspondiente al dominio *Depots* se observa que FIFO no logra una reducción significativa de la cantidad de acciones. Si se observa una reducción muy importante, mínimamente de un orden de magnitud, para los estimadores de relevancia LINR y LINR-RR. Así, *Depots* se constituye dentro del conjunto de los dominios donde la aplicación de la técnica incremental resulta muy conveniente.

En la figura 8.6 correspondiente al dominio *Satellite* se observa que FIFO se comporta de igual manera que grounding total. La reducción que observa con los estimadores de relevancia informados KRN y KRN-RR es también de alrededor de un orden de magnitud.

En la figura 8.7 correspondiente al dominio *TPP* se observa una reducción con LIFO por debajo de un orden de magnitud principalmente en las tareas de mayor tamaño, o sea, para valores más grandes de RG^{+}. Desafortunadamente no hay reducción con los estimadores de relevancia SVR y SVR-RR, lo cual, convierte al dominio *TPP* en el peor dominio, entre los dominios del benchmark, para la aplicación de nuestra técnica incremental.

En conclusión, se observa que los modelos aprendidos logran capturar en cada dominio y con bastante precisión aquellas acciones que resultan más relevantes para resolver una tarea, llevando esto a una sustancial reducción del tamaño de la tarea procesada producto de una disminución en su cantidad de acciones. Esto se observa fundamentalmente en los dominios *Agricola*,

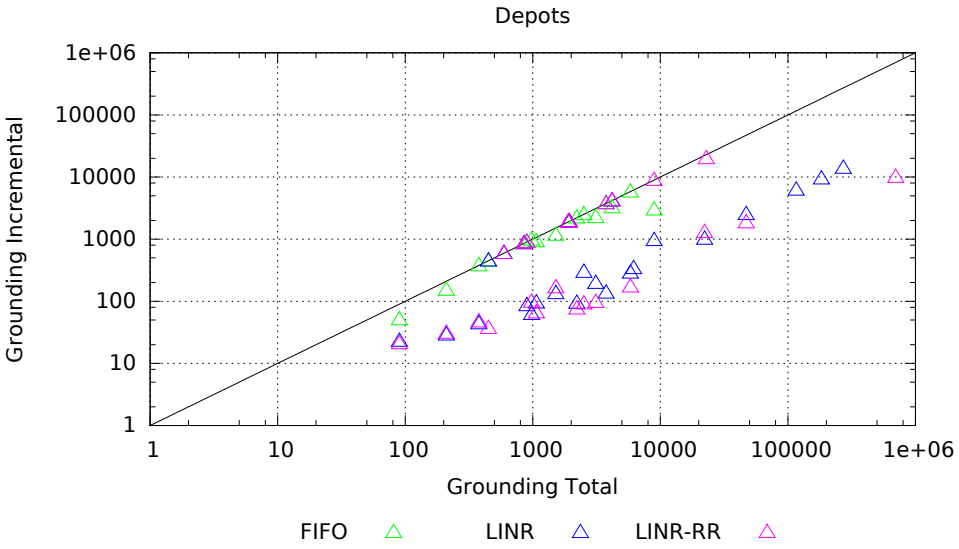


Figura 8.5: Cantidad de acciones procesadas por grounding incremental vs grounding total en el dominio *Depots*.

Blocksworld y *Caldera*. En menor medida en *Depots* y *Satellite*. Mientras en *TPP* se observa una mínima reducción con una heurística no informada (LIFO). También se observa que diferentes modelos ML actúan mejor dependiendo del dominio y esto hace difícil determinar qué modelo ML funciona mejor para la técnica incremental en general. Por ejemplo, en el dominio *Depots* la función de relevancia LINR fue la mejor, mientras KRN lo fue en el dominio *Satellite*. Además, en la mayoría de los dominios la implementación de round robin (RR) no se traduce en un cambio considerable con respecto al modelo sin round robin. A lo sumo podemos mencionar que LOGR-RR reduce las acciones procesadas levemente por encima de LOGR en el dominio *Agricola*. Pero en el dominio *Blocksworld* ocurre lo contrario, es decir, LOGR logra reducir apenas un poco más la cantidad de acciones procesadas con respecto a LOGR-RR. Por su parte, las funciones de relevancia no informadas, en la mayoría de los dominios, no reducen significativamente la cantidad de acciones procesadas. Este es el caso de FIFO en *Agricola*, *Depots* y *Satellite*. Aunque esto último no ocurre tan estrictamente con FIFO en *Caldera* y con LIFO en *TPP*.

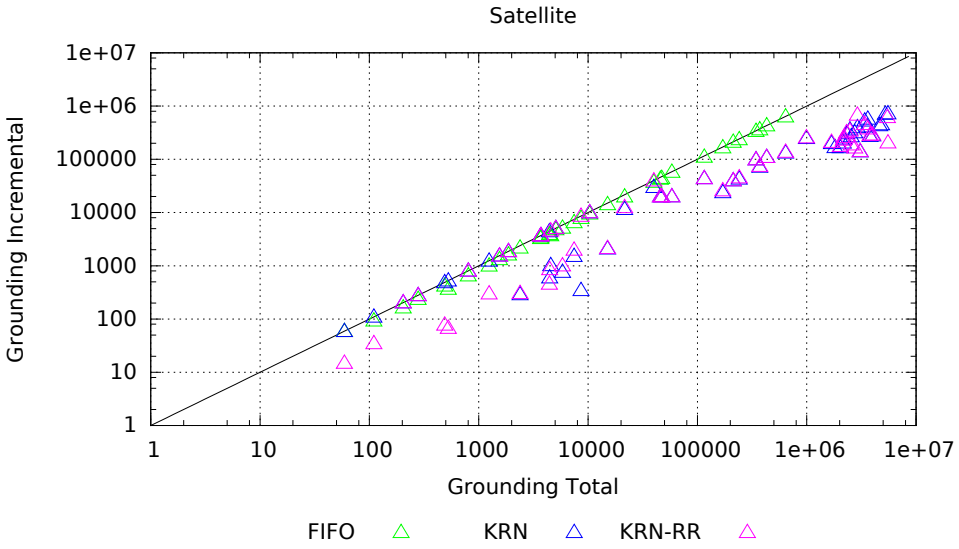


Figura 8.6: Cantidad de acciones procesadas por grounding incremental vs grounding total en el dominio *Satellite*.

8.4. Estimador SG

En esta sección presentamos una breve evaluación sobre el estimador SG definido en la ecuación 7.2 de la sección 7.1.2. Para ello, el experimento que proponemos consiste en ejecutar grounding incremental parametrizado de igual manera a como lo hicimos en la sección 8.3 con la única diferencia de que en la primera iteración del algoritmo, en lugar de procesar GG^+ acciones, procesamos SG acciones. La intención detrás de este experimento es observar cuántas iteraciones ahorramos si contamos con un modelo, en este caso obtenido por ML, que estima la cantidad de acciones suficientes a procesar para obtener un plan de la tarea. Además, este ahorro posible de iteraciones, implica que ganamos tiempo para realizar más iteraciones en busca de un plan de la tarea producto del tiempo ahorrado en las primeras verificaciones que resultaban en un fracaso tipo I (ver sección 7.1.3). Es decir, el ahorro de las primeras iteraciones cuya verificación fallaba puede ocasionar que tareas que no eran resueltas por grounding incremental, ahora son resueltas gracias a la implementación del estimador SG. Por ejemplo, supongamos que el algoritmo de grounding incremental, donde la primera iteración procesa GG^+ acciones, terminó hallando un plan de la tarea después de 7 iteraciones. Esto implica que era necesario procesar alrededor de $GG^+ + 6 \times 10.000$ acciones para hallar un plan de la tarea. Entonces, si el estimador SG devuelve un

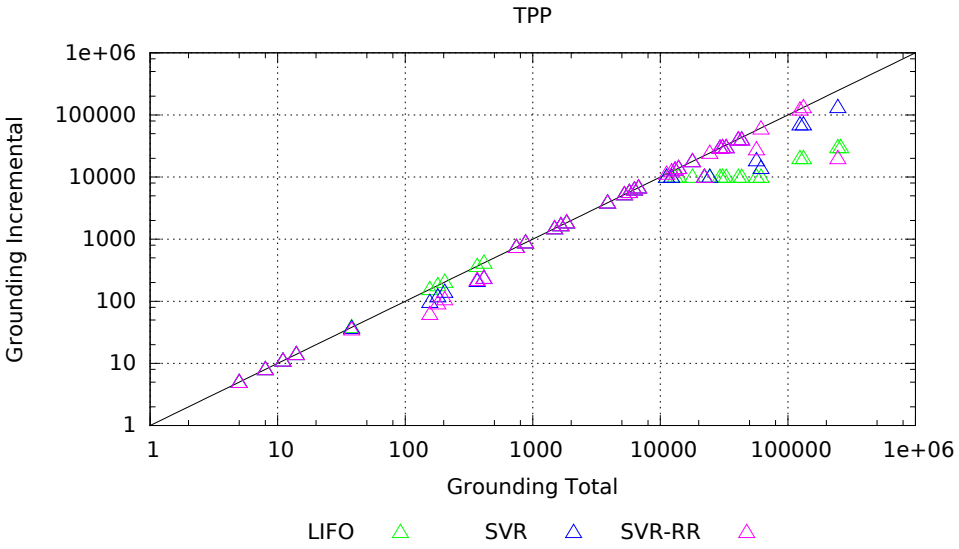


Figura 8.7: Cantidad de acciones procesadas por grounding incremental vs grounding total en el dominio *TPP*.

valor cercano a $GG^+ + 6 \times 10.000$ es muy probable que el mismo algoritmo de grounding incremental, si en la primer iteración procesa SG acciones, encuentre un plan en una sola iteración. Claramente, la cantidad de iteraciones ahorradas es directamente proporcional a la calidad de la estimación de SG. En definitiva, el experimento intenta demostrar cuantas iteraciones son ahorradas gracias a SG. Y si dicho ahorro de iteraciones, que se traduce en ahorro de tiempo, permite ahora hallar un plan cuando antes no lo hacía, tal vez, por falta de tiempo.

En el cuadro 8.5 mostramos los resultados obtenidos para los dominios *Blocksworld*, *Caldera*, *Satellite*, *TPP* al ejecutar grounding incremental donde en la primera iteración procesamos GG^+ y SG acciones, respectivamente.[1] Los mejores resultados se observan en los dominios *Caldera* y *TPP*. En el dominio *Caldera*, la utilización del estimador SG mantuvo el coverage ideal (20/20) y redujo el promedio de iteraciones (de 2.5 a 1.3). Mientras que para el dominio *TPP*, el estimador logró resolver una tarea más (de 1 a 2) y disminuyó el promedio de iteraciones (de 7.4 a 5). Por otro lado, los peores resultados se observan en los dominios *Blocksworld* y *Satellite*. En el domino *Blocksworld* se observa que el estimador SG resuelva una tarea menos (de 49 a 48). Además aumenta levemente el promedio de la cantidad de iteraciones (de 1.8 a 2). Mientras que en el dominio *Satellite* el estimador

[1] Los dominios del benchmark *Agricola* y *Depots* no fueron evaluados en esta ocasión.

		GG^+		SG	
Dominio	#	Cov	Ite	Cov	Ite
Blocksworld	50	**49**	**1.8**	48	2
Caldera	20	20	2.5	**20**	**1.3**
Satellite	25	**19**	**7.6**	15	9
TPP	20	1	7.4	**2**	**6**
$\sum$	115	**89**	19.3	85	**18.3**

Cuadro 8.5: Resultados grounding incremental con GG^+ vs SG acciones procesadas en la primer iteración. Columna "#" cantidad de tareas evaluadas, "Cov" coverage del dominio, "Ite"promedio cantidad de iteraciones. Mejor coverage y promedio de iteraciones resaltado en negrita.

SG resuelve cuatro tareas menos (de 19 a 15) pero tiene una mejora en el promedio de iteraciones (de 7.4 a 6). En conclusión, en estos dominios, si bien el comportamiento del algoritmo de grounding incremental con el estimador SG no es homogéneo, si se observa que existe una probabilidad de mejora. En particular, sobre estos cuatro dominios, en dos de ellos (50 %) conseguimos una mejora.

8.5. Baseline vs Grounding Condicional

En esta sección presentamos una breve evaluación sobre las reglas condicionales *duras* obtenidas para los seis dominios de nuestro benchmar (*Agricola*, *Blocksworld*, *Caldera*, *Depots*, *Satellite* y *TPP*). La evaluación está enfocada exclusivamente en analizar el impacto que produce la introducción de reglas condicionales duras durante el proceso de grounding. Para ello, el experimento que proponemos consiste en ejecutar grounding incremental parametrizado de igual manera a como lo hicimos en la sección 8.3 con la única diferencia de tomar como función de relevancia una heurística no informada (FIFO y LIFO), en lugar de las informadas obtenidas por ML (LINR, LOG, KRN, SVR). La intención detrás de optar por desinformar la función de relevancia se basa en que si observamos con respecto al baseline una mejora o desmejora en el grounding con reglas condicionales, deseamos que ese comportamiento esté totalmente explicado por la introducción de las reglas duras y que no esté quizás influenciado también por la calidad de la función de relevancia informada. De alguna manera LIFO y FIFO nos sirven para "desactivar" la función de relevancia no condicional. En particular, vamos a comparar el *coverage* del baseline versus el grounding incremental FIFO y LIFO sin reglas y con reglas condicionales (duras).

En el cuadro 8.6 se puede apreciar los resultados que se obtuvieron cuando introducimos reglas condicionales duras para guiar el proceso de grounding incremental. Para este experimento se generaron un extra de tareas "large" para cada dominio con respecto al experimento anterior y cuyos resultados fueron reportados en las tablas 8.1 y 8.3 (esto explica porque en la columna "#" para tareas large se puede observar un valor distinto a las dos tablas mencionadas). Se observa que el mejor resultado se obtuvo en el domino *Caldera* donde se presenta un leve incremento del coverage del 65 % (13/20) al 75 % (15/20) en las tareas IPC y un incremento importante del 32 % (26/80) al 76 % (61/80) para tareas large en FIFO-D. Otra pequeña mejora ocurre en el dominio *Depots* donde LIFO-D permitió resolver una tarea más (2/25) con respecto al baseline (1/25). Además, FIFO-D presenta el mejor coverage total 50 % (206/405), mientras que para el baseline fue de 45 % (183/405). Desafortunadamente, en los dominios *Blocksworld*, *Satellite*, *TPP* podemos decir, en general, que el resultado de implementar reglas condicionales no produce ni siquiera una leve mejora en el coverage, incluso en algunos casos se presenta una pequeña desmejora. Por ejemplo, en *TPP* tenemos un coverage de 8/25 para baseline contra 6/25. Incluso, a veces la desmejora es mayor, por ejemplo, en el dominio *Agricola* la caída del coverage fue de 4/36 a 0/36, si consideramos el caso de LIFO-D. O en *TPP* se registró una caída del coverage de 8 a 0 nuevamente para LIFO-D. Mencionamos que el mejor resultado con implementación de reglas condicionales se obtuvo para el dominio *Caldera* con FIFO-D. Por lo tanto, vamos a escoger este dominio para analizar, mediante una gráfica, la cantidad de acciones procesadas y compararlas con las obtenidas por grounding total, específicamente sobre el conjunto de tareas large (80). Para esto, cada punto de la figura **??** se corresponde con una tarea donde grounding condicional termina exitosamente, es decir, hallando un plan de la tarea. En el eje vertical tenemos la cantidad de acciones que fueron procesadas por grounding condicional y en el eje horizontal la cantidad de acciones por grounding total. Claramente, se observa una reducción muy importante en la cantidad de acciones procesadas. Esta reducción se encuentra alrededor de los dos órdenes de magnitud. Por ejemplo, aquellas tareas que tienen un valor RG^+ entre 1 millón y 10 millones de acciones , ahora mediante FIFO-D se logran resolver con menos de 100 mil acciones.

En conclusión, los resultados no son consistentes a lo largo de todos los dominios. En algunos de ellos, las reglas condicionales sirvieron para mejorar el coverage, como fue el caso de *Caldera* (de 26 a 61) y apenas una tarea en *Depots*(de 1 a 2). Pero en el resto de los dominios no fue así, inclusive en algunos casos los resultados fueron un poco peores que el baseline. Sin embargo, consideramos que estos resultados por ser producto de reglas con-

Dominio	#	Base	FIFO	FIFO-D	LIFO	LIFO-D
Agricola-IPC	20	**10**	1	9	2	2
Agricola-large	36	**4**	0	**4**	0	0
Blocksworld-IPC	35	**35**	**35**	**35**	**35**	**35**
Blocksworld-large	31	0	0	0	0	0
Caldera-IPC	20	13	14	**15**	12	10
Caldera-large	80	26	50	**61**	24	21
Depots-IPC	22	**20**	19	19	**20**	**20**
Depot-large	25	1	0	0	1	**2**
Satellite-IPC	36	**36**	35	27	**36**	30
Satellite-large	45	0	0	0	0	0
TPP-IPC	30	**30**	**30**	**30**	**30**	**30**
TPP-large	25	**8**	6	6	**8**	0
$\sum$	405	183	190	**206**	168	150

Cuadro 8.6: *Coverage obtenidos para baseline (Base); grounding incremental desinformado sin reglas (FIFO, LIFO) y con reglas condicionales duras (FIFO-D, LIFO-D). # cantidad de tareas. Mejor valor coverage resaltado en negrita.*

dicionales muy básicas o primitivas, representan un buen punto de partida. Y más aún, si consideramos que vamos a concentrar nuestros esfuerzos de los próximos meses en la generación de reglas condicionales más complejas con la expectativa de que esto derive en resultados más consistentes.

,,

,,,,

Capítulo 9

Conclusiones

La ida de subdominización surgió al observar que en gran parte de los dominios de planning solamente una fracción pequeña de las acciones eran realmente necesarias para resolver la tarea. Es muy común observar problemas con cientos de miles de acciones y con planes de a lo sumo unas cientos de ellas. Más aún, incluso en algunos dominios muy puntuales, que no forman parte de las competencias internacionales de planning (IPC), se ha observado que solo un subconjunto pequeño de esquemas de acción del dominio son necesarios para resolver gran parte de las tareas. Esto sugiere que inclusive podríamos descartar esquemas de acción en lugar de solo ciertas acciones puntuales de un esquema. A partir de esa observación, nos propusimos optimizar el proceso de generación de acciones proposicionales del planificador enfocando el proceso de grounding sobre aquellas que son realmente relevantes para conseguir una solución del problema.

En particular, nos concentramos en estudiar el proceso de grounding actualmente implementado en el planificador FD, por ser este, el framework más utilizado actualmente por la comunidad de planning. Descubrimos que este proceso consistía básicamente del cómputo de acciones relajadamente alcanzables, y por lo tanto, implementamos nuestra noción de subdominización como un cómputo parcial del mismo. Es decir, como una optimización del proceso de grounding por alcanzabilidad relajada de FD.

En el capítulo 7 presentamos los conceptos fundamentales de la técnica de subdominización que radica en dos modificaciones claves del proceso de grounding total.[1] La primer modificación consiste en cambiar el orden en que las acciones son procesadas por el algoritmo, mediante la introducción de heurísticas, obtenidas por técnicas de ML, que indican qué acciones son más relevantes para alcanzar la meta para ser priorizadas en el algoritmo por sobre

[1] Recordar que el proceso de grounding por alcanzabilidad relajada es llamado "proceso de grounding total" en los capítulos 7 y 8.

el resto de la acciones. En consecuencia, la efectividad de la técnica se basa en la capacidad de construir heurísticas para guiar el proceso de grounding que efectivamente distingan a aquellas acciones realmente relevantes para obtener una solución del problema.

Para esto tuvimos la necesidad de introducir el concepto de reglas relacionales, con el propósito de abstraer los objetos concretos que ocurren en las acciones proposicionales en busca de generalidad, y que a su vez, tengan la capacidad de capturar las características intrínsecas de aquellas acciones que resultan verdaderamente relevantes para la tarea. Esta reglas buscan una conexión entre los posibles objetos con los cuales un esquema de acción se puede instanciar, con los objetos que ocurren en el estado inicial y la meta del problema. Finalmente, estas reglas cumplen el rol de ser los atributos (feature-vector) en la abstracción de nuestras instancias de entrenamiento y de esta manera permitirnos utilizar métodos estándares de clasificación y regresión. Como ya dijimos, una mayor exactitud en la estimación de relevancia se traduce en una mayor reducción en la cantidad de acciones proposicionales de la tarea, convirtiendo la obtención de buenas reglas relacionales en el punto más clave de la técnica.

Las segunda modificación consiste en cambiar la condición de parada del algoritmo permitiendo terminar mucho antes. Por supuesto, esto es posible gracias a la heurística de relevancia siempre que esta logre procesar primero las acciones relevantes. Dicho de otra manera, una mayor calidad en las estimaciones de relevancia habilita a una terminación más temprana del algoritmo. Para esto propusimos dos alternativas. La primera es determinar una cantidad de acciones máxima a procesar limitada a los recursos computacionales (tiempo y memoria) disponibles del planificador. Esta alternativa, tiene la dificultad de que es muy difícil determinar de antemano en forma relativamente precisa ese número de acciones límite. Las segunda alternativa consistió en obtener un estimador, mediante ML, de la cantidad de acciones suficientes a procesar para hallar una plan de la tarea. Aquí, la dificultad radica en que ML implica un etapa de entrenamiento, y por ende, la selección de instancias de entrenamiento adecuadas para ello. Y esto último, puede convertirse en un incordio para un usuario común de la técnica. Esta dos modificaciones constituyen la idea fundamental del algoritmo de grounding parcial. Esto sumado a la estrategia incremental para recuperarnos de una verificación fallida de nuestra subdominización garantizando la correctitud de la misma, se convierten de esta manera en una herramienta muy útil a la hora de intentar reducir considerablemente la cantidad de acciones proposicionales que genera el proceso de grounding del planificador.

En el capítulo 8 evaluamos la técnica empíricamente en seis dominios distintos sobre tareas cuyo proceso de grounding presentaban una complejidad

considerable. Los resultados demuestran la efectividad de nuestro método. Para muchos dominios observamos un aumento considerable de la métrica *coverage*. Es decir, el grounding parcial logra resolver más tareas que el baseline, el cual, incluye un proceso de grounding total. En las gráficas observamos una reducción significativa en la cantidad de acciones procesadas por el algoritmo. Estas reducciones inclusive llegaron alcanzar los dos órdenes de magnitud en varios casos. Estas mejoras son explicadas por el hecho de que, en la mayoría de los dominios, los modelos entrenados para estimar la relevancia de las acciones fueron capaces de discriminar con precisión aquellas acciones verdaderamente relevantes, y en consecuencia, permitiendo al algoritmo terminar mucho antes que todas las acciones relajadamente alcanzables sean procesadas, y aún así, contener acciones que pertenecen a un plan de la tarea. Este ahorro significativo de acciones se traduce en una reducción en el tamaño de la tarea sobre la cual se realiza finalmente el proceso de búsqueda heurística.

Luego, evaluamos la mejora sobre la condición de parada del algoritmo considerando la estrategia de estimar, mediante ML, la cantidad de acciones suficientes a procesar para hallar un plan de la tarea (estimador SG). El experimento realizado tenía el propósito de analizar si el ahorro de las primeras iteraciones en el algoritmo incremental, producto de contar con la información provista por el estimador SG, deriva en un aumento de las tareas resueltas (coverage). Los resultados arrojaron una pequeña mejora en los dominios *Caldera* y *TPP*. Por último, reportamos los resultados correspondientes a la implementación de reglas condicionales duras. El objetivo del experimento es verificar empíricamente si la ocurrencia de ciertas acciones pueden alterar efectivamente la relevancia de otras. El resultado más rutilante lo obtuvimos en el dominio *Caldera*, donde logramos resolver más del doble de las tareas que con el baseline (de 26 a 61). Esto se explica en el hecho de que la cantidad de acciones procesadas se redujo en esta ocasión en dos órdenes de magnitud aproximadamente. En definitiva, los resultados arrojados por los distintos experimentos evidencian la utilidad práctica de la técnica.

La idea de subdominización, implementada en nuestro caso como un proceso de grounding parcial en relación con el actual algoritmo de grounding del planificador FD, puso de manifiesto el hecho de que muy pocas acciones, a lo sumo unas cientos de ellas, son necesarias para resolver un problema. De esta manera, subdominización, por sus fundamentos y sus resultados empíricos expuestos en este trabajo, se convierte en una opción muy prometedora para atacar el problema de grounding. Más precisamente, en aquellas tareas donde el proceso de grounding total es inviable por su magnitud dado los limitados recursos computacionales que puede tener un planificador.

Parte IV

Conclusiones

Capítulo 10

Conclusiones Generales

En esta tesis hemos presentado y estudiado el problema de grounding para planning clásico, comprendiendo la dimensión del problema que representa. Por su naturaleza exponencial (con respecto a la aridad de los esquemas de acción y predicados de los dominios) y su naturaleza escalable (con respecto a la cantidad de objetos del entorno modelado), resulta de vital importancia encontrar métodos que ayuden a mejorar, mediante tiempos de respuesta razonables, el proceso de grounding en los planificadores modernos. Todo esto con el objetivo final de acercar la utilización de planificadores automáticos a la resolución de problemas concretos, que se presentan en distintos ámbitos de la vida real, tales como, aplicaciones industriales, comerciales, financieras, de logística y hasta académicas.

Sabemos que el proceso de grounding es una componente dentro del planificador que se encarga de, a partir de una representación de alto nivel (de naturaleza esquemática), y en consecuencia, muy amigable para el diseñador del dominio, computar una representación de más bajo nivel (de naturaleza proposicional) del problema que resulta ventajosa para la aplicación directa de efectivos algoritmos de búsqueda basados en heurísticas. Es por esto que es clave, que el proceso de grounding finalice y retorne la tarea proposicional más compacta posible con tiempos de cómputo aceptables convirtiendo al planificador en una herramienta concreta para ser utilizada en la práctica.

Para hacer frente a este desafío, propusimos dos alternativas que intentan optimizar el tamaño de la representación proposicional del problema devuelto por el proceso grounding. Esta representación es luego utilizada por el planificador para llevar adelante un proceso de búsqueda con el afán de hallar finalmente una solución del mismo, Ambas técnicas actúan en diferentes puntos del pipeline implementado en el planificador, permitiendo esto incluso la combinación de ellas.

Nuestra primer técnica propuesta *Action Schema Splitting* opera directa-

mente sobre la representación PDDL de la tarea, reemplazando los esquemas originales del dominio por sub-esquemas, en forma cuidadosa e inteligente, para preservar los planes de la tarea original en una relación uno a uno. De esta manera, logramos reducir significativamente la aridad de los esquemas del dominio original pero con la contra parte de aumentar la distancia a la meta, sabiendo que esto último, es costoso a la hora de la búsqueda heurística.

A partir de los resultados experimentales observados, podemos asegurar que splitting genera una disminución considerable del grounding, salvo en algunos casos muy puntuales que se corresponden con splittings muy agresivos, es decir, con un nivel de granularidad muy elevado obtenido mediante valores de γ cercanos a 0. Sin embargo, para valores menos extremos de γ, los splitting homogéneamente optimizan el grounding.

Incluso, en dominios donde se presentaban serios problemas de grounding, se observó que la reducción en el número de instancias generadas llegaba a alcanzar los dos órdenes de magnitud, demostrando que splitting con un nivel de granularidad balanceado (es decir, sin introducir un exceso de sub-acciones) se convierte en una herramienta valiosa de optimización del grounding para este tipo de dominios.

Además, observamos que el valor de γ impacta en forma distinta de acuerdo a cada dominio. Es decir, existen dominios que son más sensibles al valor de γ que otros. Para ciertos dominios con valor de γ cercano a 1, conseguimos inmediatamente un splitting suave del mismo, es decir, de pocas sub-acciones, mientras que en otros dominios necesitamos un valor mucho más alejado de 1 para causar un efecto similar. En conclusión, con respecto al valor de γ debe ser definido para cada dominio particular de acuerdo al nivel de granularidad que se busca.

En contra parte, la cantidad de sub-acciones generadas por el splitting deteriora el rendimiento de la búsqueda. Aunque esto no debe sorprender, ya que, sabemos que aumentar la profundidad de un espacio de búsqueda tiene un impacto mayor que aumentar la anchura del mismo. Claramente, la técnica de splitting reduce el factor de branching pero con el costo de aumentar el factor de profundidad.

Es un interrogante abierto si el aumento de estados expandidos en el splitting es producto de la introducción de estados intermedios como consecuencia del reemplazo de acciones por sub-acciones ó es producto de que el splitting afecta los valores heurísticos (h-value) de los estados. Intuitivamente, nos inclinamos por la segunda hipótesis, ya que, el splitting claramente no preserva los planes de la tarea relajada, producto de que no contamos con los efectos negativos de la lista *del* de cada acción. Por lo tanto, resulta afectado el sistema de tokens implementado por las sub-acciones del splitting. De esta manera, en el dominio relajado no podemos asegurar que un splitting es eje-

cutado en bloque y consistentemente en la tarea relajada. Estas propiedades solamente son aseguradas en el problema original, o sea, en el problema sin relajar. Es muy probable que esto cause un cambio del valor heurístico de los estados, ya que, recordar que la heurística para computar sus estimaciones se basa en la distancia del estado a la meta en la tarea relajada. En otras palabras, el splitting no preservaría la decisiones tomadas por el algoritmo de búsqueda que hubiesen ocurrido en el caso del dominio original.

En síntesis, la técnica de splitting debe ser quirúrgicamente aplicada, principalmente en aquellos dominios que presentan verdaderos problemas de grounding, a tal punto que el proceso de grounding total, es decir, por alcanzabilidad relajada, no puede terminar de generar la representación proposicional de la tarea. Además, es requisito del éxito de la técnica que el problema de grounding sea producto de la presencia de esquemas de acción en el dominio con largas interfaces (aptas para ser divididas por nuestra técnica) y gran cantidad de objetos definidos en la tarea.

Nuestra segunda técnica propuesta *Subdominización* directamente es un modificación del proceso de grounding actual usado en la mayoría de las herramientas actuales de planning. En particular, hemos modificado el algoritmo utilizado por planificador FD, que es actualmente el modelo en el que se basan la mayoría de los planeadores existentes.

Más precisamente, modificamos el orden en que las acciones son procesadas por el algoritmo intentando priorizar aquellas que resultan más relevantes para resolver la tarea. Debemos también modificar la condición de parada del algoritmo, para permitir terminar mucho antes de que todas las acciones relajadamente alcanzables sean procesadas, siempre que hayamos sido capaces de procesar acciones que pertenecen a algún plan de la tarea.

Mientras más precisos seamos para predecir la relevancia de una acción, el proceso de grounding puede completarse en forma temprana, retornando una representación proposicional de la tara con mucho menos acciones y que aún conserva entre ellas un subconjunto de acciones que pertenecen a algún plan. Por lo tanto, el éxito de la técnica radica principalmente en la calidad de la estimación del nivel de relevancia de las distintas acciones.

Las estimaciones se obtienen mediante modelos de machine learning a partir de tareas del dominio donde el proceso de grounding total y búsqueda finaliza exitosamente. Los modelos de machine learning elegidos deben ser lo suficientemente complejos para ser capaces de aprender las características intrínsecas que llevan a una acción a ser relevante, pero al mismo tiempo ser rápidos en su respuesta, ya que, el proceso de grounding llamará al modelo estimador permanentemente.

Para esto introducimos la noción de reglas relacionales que vinculan los

objetos con los cuales un esquema de acción es instanciado con los facts que ocurren en el estado inicial y la meta de la tarea. Estas reglas relacionales terminan siendo los atributos que constituyen el vector de atributos que representa cada instancia de entrenamiento. De esta forma, logramos abstraer los objetos concretos que ocurren en las acciones presentes en las instancias de entrenamiento.

En este trabajo proponemos un primer alternativa para la generación de reglas relacionales empleando una estrategia de fuerza bruta. Es aquí donde surge otro desafío interesante: tratar de mejorar estas reglas relacionales buscando siempre que las mismas brinden una mayor ganancia de información o entropía sobre que tan relevante es una acción.

Aún cuando el modelo estimador de relevancia sea bastante preciso, eso no garantiza que las acciones señaladas como más relevantes contengan efectivamente un plan de la tarea. Es por eso, que computada una subdominización es necesario realizar una verificación de la misma, que consiste en chequear si contiene las acciones correspondiente a un plan. Pero esto último, sólo se puede lograr si ejecutamos una búsqueda. En caso de no hallar un plan, no podemos garantizar que la subdominización es correcta. Por lo tanto, implementamos una estrategia incremental permitiendo agrandar el tamaño de la tarea proposicional considerando una cantidad extra de acciones. Así sucesivamente hasta conseguir una verificación exitosa que asegure la correctitud de la subdominización computada hasta ese momento, o converger en un grounding total.

Los experimentos realizados demostraron la efectividad de la técnica a la hora de conseguir, en múltiples ocasiones, una tarea proposicional con muchas menos acciones, y aún así, con la cantidad suficiente como para contener un plan de la tarea convirtiendo a la técnica en una herramienta muy útil sobre todo en aquellos problemas donde el proceso de grounding agotaba los recursos computacionales del planificador impidiendo incluso el comienzo de la etapa de búsqueda.

Más tarde, y retomando el propósito de mejorar las estimaciones de relevancia, propusimos reglas condicionales que intentan capturar las dependencias entre distintas acciones. Esto surgió de la observación de que ciertas acciones ocurren siempre que otras sean ejecutadas previamente. Es decir, la ocurrencia de una acción puede condicionar la ocurrencia de alguna otra. Dicho de otro modo, la ocurrencia de ciertas acciones pueden modificar dinámicamente el nivel de relevancia de otras. Para detectar estas dependencias propusimos la idea de reglas condicionales duras y blandas que difieren entre sí en el nivel de ligamiento de las variables que ocurren en las reglas. En este trabajo evaluamos las subdominizaciones obtenidas producto de reglas duras, y considerando que es nuestra primer aproximación al concepto de

reglas condicionales ha demostrado tener resultados prometedores.

Por otro lado, todavía resta proponer una alternativa para el tratamiento de variables libres que ocurren en reglas blandas para poder proceder a su evaluación.

Es importante mencionar que la idea de subdominización va más allá de la propuesta de grounding parcial respecto del algoritmo de alcanzabilidad relajada actualmente implementado en FD. Resulta interesante estudiar si es posible conseguir nuevas formas de subdominización sin tener que ser estrictamente una proyección de la tarea relajada.

En definitiva, ambas técnicas son, a nuestro entender, muy valiosas desde el punto de vista práctico y teórico. En lo práctico, constuyen algoritmos concretos y efectivos para atacar el problema de grounding, incluso en dominios que modelan problemas complejos de la vida real. Su importancia teórica radica, entre otras cosas, en que el proceso de grounding nunca había sido un tema central de investigación del área. Siempre fue relegado por investigaciones orientadas a la mejora del proceso de búsqueda heurística. Es por esto que tenemos la ambición de que este trabajo sea un punto de partida e inspiración para futuras investigaciones sobre cómo hacer grounding por parte de la comunidad de planning.

Muchas gracias!

"Si antepones la igualdad por encima de la libertad, es muy posible que consigas poco de ambas. Si antepones la libertad por encima de la igualdad, es muy posible que consigas mucho de ambas"
Milton Friedman

Capítulo 11

Trabajo Futuro

Nuestras investigaciones en planning están direccionadas a trabajar puntualmente en el proceso de grounding de un planificador. Esta componente dentro del planificador es de vital importancia para la obtención de soluciones en forma automática o algorítmica de cualquier problema que se puede representar en términos de una situación inicial, acciones posibles a ejecutar y una meta a alcanzar. En otras palabras, la componente de grounding de un planificador es la primer gran componente de un pipeline de trabajo en la búsqueda de un *solver general* de problemas. Esta componente permite, desde una descripción de alto nivel, y por ende, más flexible y compacta para describir un problema, obtener una representación de más bajo nivel más conveniente para aplicar sobre ella poderosos y eficientes algoritmos para la obtención de soluciones en forma automática. Por todo esto, nuestra dos técnicas de optimización del grounding se convierten en pioneras para enfrentar complicaciones que pueden surgir desde esta componente.

Nuestros esfuerzos en el futuro estarán enfocados fundamentalmente en descubrir diferentes alternativas para afrontar el problema de grounding. Así como también mejorar y estudiar muchos de los aspectos no cubiertos en este trabajo con respecto a nuestras dos técnicas ya presentadas en esta tesis.

Por ejemplo, dado el efecto no deseado que genera el splitting sobre el espacio de estados de una tarea de planning, resulta interesante estudiar el motivo que explica tal aumento de estados para luego intentar atenuarlo. Nuestra intención es estudiar si el aumento se produce sólo por la introducción de estados "intermedios" producto del reemplazo de acciones por sub-acciones ó si se produce porque el splitting modifica los valores computados por la heurística alterando el orden en que los estados son seleccionados por la misma para ser expandidos. Una forma de ver esto es comparar si los planes del dominio spliteado son los mismos planes que se obtienen en el dominio original. Si en general, lo son, entonces el aumento de estados ex-

pandidos está explicado en la aparición de los estados intermedios producto de ejecutar un splitting en lugar de una acción y si es este el caso, quizás no haya mucho por hacer. Si no lo son, entonces el splitting está afectando las decisiones de la heurística. En este último caso, resulta interesante lograr de alguna manera que el splitting además de preservar los planes como ya lo hicimos, también preserve la heurística de búsqueda.

Otro interrogante relacionado con atenuar los efectos del splitting sobre la búsqueda es si realmente resulta necesario computar la heurística en los estados intermedios del splitting. Tal vez, podemos informar al algoritmo de búsqueda cuál es la siguiente sub-accion a ejecutar sin tener que llamar a la heurística, lo cual significaría un ahorro importante de tiempo de cómputo en la etapa de búsqueda.

Otra cuestión a estudiar consiste en adaptar splitting para que tenga en cuenta información de la tarea a la hora de dividir los esquemas de un dominio. Notar que hasta este momento la técnica se basa exclusivamente en la estructura interna de cada esquema para computar un splitting. No se utiliza ninguna información proveniente del estado inicial, la meta ó los objetos definidos del entorno modelado. Esto llevaría, tal vez, a un splitting más preciso dada una determinada tarea pero con la pérdida de generalidad, ya que, tendríamos para un mismo dominio diferentes dominios spliteados para cada tarea particular. Esta idea también aplica para la noción de spliteabilidad que actualmente está basada solo en características estructurales propias de cada esquema y no considera ninguna información proveniente de la tarea.

A partir de las citas a nuestra publicación sobre splitting [Areces *et al.*, 2014] sabemos de la existencia de dominios que intentan modelar problemas reales de ciertas disciplinas con el fin de utilizar el "state-of-the-art" de los panificadores actuales para obtener soluciones.

Este es el caso del domino "*Chemical*" [Masoumi *et al.*, 2015] que modela un conjunto de posibles reacciones químicas a efectuar con el fin de conseguir una molécula deseada de cierto tipo. Este dominio presentan serias dificultades en el proceso de grounding, lo cual, lo convierte en un dominio muy interesantes para testear nuestra técnica. En este sentido, debemos intentar conseguir más de este tipo de dominios para evaluar la técnica de splitting y alejarnos progresivamente de los testings sobre dominios IPC.

En definitiva, con respecto a splitting consideramos que los esfuerzos futuros deben concentrarse principalmente en atenuar el impacto que produce el splitting en la búsqueda, fundamentalmente en dominios que resultan más adecuados para ser optimizados vía splitting. Si logramos esto, convertiremos la técnica en una herramienta básica y primordial dentro del área.

En relación a subdominización, nuestros planes más cercanos se enfocan en la generación de reglas condicionales más expresivas que realmente logren

capturar las dependencias entres las acciones de una tarea.

En particular, queremos proponer una solución al problema de la instanciación de variables libres en reglas condicionales.

Otra cuestión a estudiar es si podemos construir estimadores de relevancia basados en heurísticas tradicionales en lugar de modelos de machine learning, que sean tan informadas como estos últimos. Esto llevaría a evitar la etapa de entrenamiento del algoritmo. En muchas ocasiones, la generación de tareas adecuadas para el entrenamiento puede ser costosa.

Otro aspecto interesante para investigar consiste en utilizar, de alguna manera, la información ya generada sobre la tarea relajada por el proceso de grounding para que sea reutilizada por la etapa de búsqueda. Notar que el proceso de grounding, al fin y al cabo, realiza una búsqueda heurística sobre la tarea relajada. Mientras que el proceso de búsqueda del planificador realiza una búsqueda sobre la tarea no relajada pero apoyandosé en una heurística que utiliza la tarea relajada para computar sus estimaciones. Por lo tanto, parece tener sentido tratar de conectar lo ocurrido con la tarea relajada en tiempo de grounding con la heurística en tiempo de búsqueda. Intuitivamente, parece que estamos haciendo dos cosas muy similares que podrían unificarse, con el consecuente ahorro de tiempo que eso significaría.

Por otro lado, observamos que el proceso de grounding parcial presentado en este trabajo, es sólo un caso particular de la idea de subdominización que es más amplia y general. Por lo tanto, resulta interesante encontrar nuevas formas para obtener, a partir de la representación PDDL de la tarea, una representación proposicional más pequeña y conveniente para la búsqueda. Es decir, no necesariamente toda subdominización debe ser obtenida o pensada como un cómputo parcial del proceso de grounding por alcanzabilidad relajada. En definitiva, podemos ser más creativos a la hora de computar un subconjunto de acciones lo más pequeño posible pero que aún contenga las acciones pertenecientes a un plan de la tarea.

Otra linea de investigación interesante es hallar la forma de combinar ambas técnicas propuestas en este trabajo, con el propósito de obtener una técnica mixta que combine las ventajas de ambas y neutralizando sus efectos negativos. En el caso del splitting nos referimos al aumento del espacio de estados y en el caso de subdominización a su incompletitud, que derivó en una estrategia incremental que afecta la eficiencia de la técnica.

Notar que esta combinación implicaría un salto entre diferentes niveles de abstracción, ya que, el splitting opera a nivel de esquemas de acción y subdominización lo hace a nivel de acciones proposicionales. Por este motivo, en una primer instancia parece más fácil pensar esta combinación como una composición de operadores que actúan secuencialmente sobre el problema. Primero aplicaríamos splitting y luego aplicaríamos subdominización a su

resultado.

Parece más difícil considerar la composición de los operadores al revés. Sin embargo, tal vez, podemos definir la noción de subdominización a nivel de esquemas de acción para lograr la compatibilidad con el nivel de abstracción de splitting. Esto parece tener sentido, ya que se ha observado, por ejemplo, en el dominio *Chemical* que gran parte de los esquemas de acción resultan totalmente descartables para muchas de las tareas propuestas por los diseñadores del dominio.

Apéndice A

Apéndice Dominios

Este apéndice presenta una breve descripción y las principales características de los diferentes dominios que forman parte del benchmark utilizado en los capítulos de evaluación 5 y 8 de cada técnica, respectivamente. Para cada dominio se presenta una breve descripción del problema que modela, sus principales atributos (cantidad de esquemas, máxima interfaz, etc) y un cuadro con las principales características de cada uno de sus esquemas de acción. La tabla A.1 muestra un sumario de las características por dominio.

Dominios	**Tipo**	**Esq.**	**Pred.**	**AvgInt.**	**MaxInt.**	**Splty.**
Agricola	ADL	22	33	4.77	8	0.90
Blocksworld	STRIPS	3	3	2.33	3	0.41
Caldera	ADL	8	31	5.5	11	0.78
Depots	STRIPS	5	6	3.8	4	0.48
Satellite	STRIPS	5	13	2.8	4	0.48
Tpp	STRIPS	4	8	6	7	0.56
Freecell	STRIPS	10	12	4.9	7	0.80
Pipesworld	STRIPS	4	16	10.5	12	0.62
Zenotravel	STRIPS	5	5	4.20	6	0.57
Scanalyzer	STRIPS	4	7	6.00	8	0.57
Hiking	ADL	7	9	4.57	6	0.55
Cavediving	ADL	8	17	3.38	5	0.54

Cuadro A.1: Sumario de las estadísticas principales de los dominios utilizados en los experimentos de los capítulos de evaluación 5 y 8. Columna “Esq” cantidad de esquemas, “Pred” cantidad de predicados, “AvgInt” promedio de la interfaces, “MaxInt” máxima interfaz y “Splty” valor de spliteabilidad.

A.1. Agricola

Este dominio se basa en el conocido juego de mesa Agricola, el cual consiste de una granja y trabajadores con el objetivo de aumentar la producción. Cada jugador tiene asignado un grupo inicial de trabajadores. En cada turno, un jugador debe seleccionar acciones para cada trabajador. Algunas de estas acciones sirven para obtener ciertos recursos como alimentos, madera, animales, etc. Otras acciones aumentan el número de trabajadores lo cual permite luego al jugador correspondiente realizar más acciones por turno. Pero esto al mismo tiempo aumenta la cantidad de alimentos consumida por los trabajadores, lo que puede conducir a un "dead ends". Luego de una cantidad fija de turnos, vence aquel jugador que consiguió una mayor producción. Este dominio cuenta con:

- cantidad de esquemas de acción: 22
- cantidad de predicados: 33
- promedio tamaño de interfaz: 4.77
- máximo tamaño de interfaz: 8
- valor de spliteabilidad: 0.90

En la la tabla A.2 presentamos los esquemas de acción del dominio con su correspondiente tamaño de interfaz (Int), cantidad de precondiciones atómicas (Pre), cantidad de efectos atómicos (Eff), máxima interfaz entre precondiciones y efectos atómicos (At), valor de spliteabilidad (Splty).

Agricola						
	Esquema	**Int**	**Pre**	**Eff**	**At**	**Splty**
1	COLLECT_COOK_ANIMAL	8	9	6	2	0.90
2	COLLECT_ANIMAL	6	7	4	1	0.87
3	FAMILY_GROWTH	7	8	4	1	0.86
4	TAKE_FOOD	6	7	5	2	0.85
5	AG__HARVEST_COLLECTING_VEG	6	8	4	1	0.85
6	COLLECT_RESOURCE	6	7	4	2	0.85
7	AG__HARVEST_COLLECTING_FROMOVEN	6	9	4	1	0.85
8	TAKE_GRAIN	5	5	4	1	0.84
9	BUILD_FENCES	5	6	4	1	0.84
10	AG__HARVEST_BREEDING_ANIMAL	5	7	3	1	0.84
11	IMPROVE_HOME	5	8	6	0	0.84
12	SOW	5	8	5	1	0.84
13	BUILD_ROOM	6	10	7	1	0.81
14	AG__HARVEST_FEED	6	7	7	3	0.79
15	PLOW_FIELD	4	6	4	0	0.78
16	AG__FINISH_STAGE	5	7	6	2	0.73
17	AG__HARVEST_COLLECT_END	2	3	2	1	0.67
18	AG__HARVEST_BREED_END	2	3	2	1	0.67
19	AG__FINISH_ROUND_RENEW	2	1	18	1	0.67
20	AG__FINISH_ROUND_BACKHOME	2	4	4	0	0.67
21	AG__FINISH_ROUND_BACKHOME_WITHCHILD	3	5	7	0	0.65
22	AG__ADVANCE_ROUND_NORMAL	3	4	3	2	0.41

Cuadro A.2: Esquemas de acción del dominio *Agricola* ordenados por valor de spliteabilidad.

A.2. Blocksworld

Este dominio consiste de un conjunto de bloques, una mesa (y en algunas versiones una mano robótica). Los bloques pueden estar encima de otros bloques o sobre la mesa. Un bloque que no tiene otro bloque encima, se considera "clear". La mano robótica puede sostener un bloque o estar vacía. El objetivo es encontrar un plan para pasar de una configuración de bloques inicial dada a una configuración de bloques final. Este dominio cuenta con:

- cantidad de esquemas de acción: 3
- cantidad de predicados: 3
- promedio tamaño de interfaz: 2.33
- máximo tamaño de interfaz: 3
- valor de spliteabilidad: 0.41

En la la tabla A.3 presentamos los esquemas de acción del dominio con su correspondiente tamaño de interfaz (Int), cantidad de precondiciones atómicas (Pre), cantidad de efectos atómicos (Eff),máxima interfaz entre precondiciones y efectos atómicos (At), valor de spliteabilidad (Splty).

Blocksworld						
	Esquema	**Int**	**Pre**	**Eff**	**At**	**Splty**
1	MOVE-B-TO-B	3	3	4	2	0.41
2	MOVE-B-TO-T	2	2	3	2	0.00
3	MOVE-T-TO-B	2	3	3	2	0.00

Cuadro A.3: Esquemas de acción del dominio *Blocksworld* ordenados por valor de spliteabilidad.

A.3. Caldera

Este dominio modela un problema de ciber-seguridad sobre redes de computadoras. La red consiste de una cierta cantidad de servidores, usuarios y credenciales. Este dominio cuenta con :

- cantidad de esquemas de acción: 8
- cantidad de predicados: 31
- promedio tamaño de interfaz: 5.5
- máximo tamaño de interfaz: 11
- valor de spliteabilidad: 0.78

En la la tabla A.4 presentamos los esquemas de acción del dominio con su correspondiente tamaño de interfaz (Int), cantidad de precondiciones atómicas (Pre), cantidad de efectos atómicos (Eff), máxima interfaz entre precondiciones y efectos atómicos (At), valor de spliteabilidad (Splty).

Caldera						
	Esquema	**Int**	**Pre**	**Eff**	**At**	**Splty**
1	GET_ADMIN	3	3	2	1	0.78
2	GET_COMPUTERS	2	2	2	1	0.67
3	CREDS	3	5	1	3	0.65
4	NET_TIME	3	2	6	1	0.65
5	NET_USE	11	22	1	4	0.63
6	WMIC	11	24	1	4	0.63
7	SMB_COPY	8	18	1	3	0.61
8	GET_DOMAIN	3	6	1	1	0.41

Cuadro A.4: Esquemas de acción del dominio *Caldera* ordenados por valor de spliteabilidad.

A.4. Depots

Este dominio modelo un sistema de almacenamiento de cajas que son transportadas a través de camiones a distintos depósitos. El dominio cuenta con acciones de carga y descarga de cajas por medio grúas disponibles en los diferentes depósitos. Las cajas son apiladas sobre pallets disponibles y la meta consiste depositarlas cajas en sus correspondiente depósitos. Este dominio cuenta con:

- cantidad de esquemas de acción: 5
- cantidad de predicados: 6
- promedio tamaño de interfaz: 3.8
- máximo tamaño de interfaz: 4
- valor de spliteabilidad: 0.48

En la la tabla A.5 presentamos los esquemas de acción del dominio con su correspondiente tamaño de interfaz (Int), cantidad de precondiciones atómicas (Pre), cantidad de efectos atómicos (Eff), máxima interfaz entre precondiciones y efectos atómicos (At), valor de spliteabilidad (Splty).

Depots						
	Esquema	**Int**	**Pre**	**Eff**	**At**	**Splty**
1	LIFT	4	5	6	2	0.48
2	LOAD	4	3	3	2	0.48
3	UNLOAD	4	4	3	2	0.48
4	DROP	4	4	6	2	0.45
5	DRIVE	3	1	2	2	0.41

Cuadro A.5: Esquemas de acción del dominio *Depots* ordenados por valor de spliteabilidad.

A.5. Satellite

Este dominio es un primero modelo del problema de programación de observación satelital. El problema completo implica el uso de uno o más satélites para hacer observaciones, recopilar datos y descender los datos a una estación terrestre. Los satélites están equipados con diferentes instrumentos, cada uno con diferentes características en términos de objetivos de calibración, producción de datos, consumo de energía y requisitos para el calentamiento y enfriamiento. Los satélites pueden apuntar a diferentes objetivos. Puede haber restricciones sobre qué objetivos son accesibles para cada satélites debido a las capacidades de oclusión y rotación. Los instrumentos generan datos que deben almacenarse en el satélite y posteriormente ser descargados cuando se abre una ventana de oportunidad de comunicación con una estación terrestre. Las ventanas de comunicación son fijas. Hay dificultades adicionales, como la gestión de la energía, el uso de la energía solar y el mantenimiento de las temperaturas operativas durante los períodos de sombra. Sin embargo, muchas de estas dificultades fueron descartadas, ya que, son difíciles de expresar en PDDL. La meta consiste en hallar la cobertura (más eficiente) de las observaciones dadas las capacidades de los satélites. Este dominio cuenta con:

- cantidad de esquemas de acción: 5
- cantidad de predicados: 13
- promedio tamaño de interfaz: 2.8
- máximo tamaño de interfaz: 4
- valor de spliteabilidad: 0.48

En la la tabla A.6 presentamos los esquemas de acción del dominio con su correspondiente tamaño de interfaz (Int), cantidad de precondiciones atómicas (Pre), cantidad de efectos atómicos (Eff), máxima interfaz entre precondiciones y efectos atómicos (At), valor de spliteabilidad (Splty).

Satellite						
	Esquema	**Int**	**Pre**	**Eff**	**At**	**Splty**
1	TAKE_IMAGE	4	6	1	2	0.48
2	CALIBRATE	3	4	1	1	0.34
3	TURN_TO	3	2	2	2	0.34
4	SWITCH_ON	2	2	3	1	0.00
5	SWITCH_OFF	2	2	2	1	0.00

Cuadro A.6: Esquemas de acción del dominio *Satellite* ordenados por valor de spliteabilidad.

A.6. TPP

El dominio TPP es una generalización conocida del problema del vendedor viajante (travelling salesman problem). Dada una lista de productos y la exitencia de ciertos mercados, donde cada uno de ellos cuenta con una cantidad limitada de productos a un precio conocido. El problema consiste en seleccionar un subconjunto de los mercados para que se pueda comprar una demanda dada de cada producto, minimizando el costo del itinerario y el costo de compra. Este dominio cuenta con:

- cantidad de esquemas de acción: 4
- cantidad de predicados: 8
- promedio tamaño de interfaz: 6
- máximo tamaño de interfaz: 7
- valor de spliteabilidad: 0.56

En la la tabla A.7 presentamos los esquemas de acción del dominio con su correspondiente tamaño de interfaz (Int), cantidad de precondiciones atómicas (Pre), cantidad de efectos atómicos (Eff), máxima interfaz entre precondiciones y efectos atómicos (At), valor de spliteabilidad (Splty).

Tpp						
	Esquema	**Int**	**Pre**	**Eff**	**At**	**Splty**
1	UNLOAD	7	5	4	3	0.56
2	BUY	7	5	4	3	0.55
3	LOAD	7	5	4	3	0.54
4	DRIVE	3	2	2	2	0.34

Cuadro A.7: Esquemas de acción del dominio *TPP* ordenados por valor de spliteabilidad.

A.7. Freecell

Este dominio modela el juego de cartas solitario popularizado por el sistema operativo Windows, en el cual, se colocan n columnas de cartas sobre la mesa: la primera con 1 carta, la segunda con 2 cartas, así sucesivamente hasta la n-ésima columna. La carta superior de cada columna queda hacia arriba. El jugador mueve una carta de una columna a otra siempre que la carta a mover es de un número sucesivo y de color contrario a la carta superior de la columna destino. Si la carta superior de una columna está hacia abajo, inmediatamente debe revelarse. La meta consiste en ordenar todas las cartas desde "A" hasta "K" de cada palo en una columna diferente. Este dominio cuenta con:

- cantidad de esquemas de acción: 10
- cantidad de predicados: 12
- promedio tamaño de interfaz: 4.9
- máximo tamaño de interfaz: 7
- valor de spliteabilidad: 0.80

En el cuadro A.8 presentamos los esquemas de acción del dominio con su correspondiente tamaño de interfaz (Int), cantidad de precondiciones atómicas (Pre), cantidad de efectos atómicos (Eff), máxima interfaz entre las precondiciones y efectos atómicos (At), valor de spliteabilidad (Splty).

Freecell						
	Esquema	**Int**	**Pre**	**Eff**	**At**	**Splty**
1	SENDTOFREE-B	5	6	7	2	0.80
2	NEWCOLFROMFREECELL	5	5	7	2	0.80
3	SENDTOHOME-B	7	10	6	2	0.76
4	HOMEFROMFREECELL	7	9	5	2	0.76
5	SENDTONEWCOL	4	4	5	2	0.70
6	SENDTOFREE	4	4	6	2	0.70
7	COLFROMFREECELL	4	5	6	2	0.70
8	MOVE-B	4	6	5	2	0.70
9	SENDTOHOME	6	8	5	2	0.57
10	MOVE	3	4	4	2	0.41

Cuadro A.8: Esquemas de acción del dominio *Freecell* ordenados por valor de spliteabilidad.

A.8. Pipesworld

Este dominio modela un problema de red oleoductos obedeciendo varias restricciones, como la compatibilidad del producto, las restricciones de tanques y los plazos de los objetivos. Un aspecto interesante del dominio es que, si uno inserta algo en un extremo de un segmento de canalización (pipe), algo potencialmente completamente diferente sale en el otro extremo. Esto da lugar a varios fenómenos sutiles que pueden surgir en la creación de un plan. Este dominio cuenta con:

- cantidad de esquemas de acción: 4
- cantidad de predicados: 16
- promedio tamaño de interfaz: 10.5
- máximo tamaño de interfaz: 12
- valor de spliteabilidad: 0.62

En el cuadro A.9 presentamos los esquemas de acción del dominio con su correspondiente tamaño de interfaz (Int), cantidad de precondiciones atómicas (Pre), cantidad de efectos atómicos (Eff), máxima interfaz entre las precondiciones y efectos atómicos (At), valor de spliteabilidad (Splty).

Pipesworld						
	Esquema	**Int**	**Pre**	**Eff**	**At**	**Splty**
1	POP	12	13	12	2	0.62
2	PUSH	12	14	12	2	0.62
3	PUSH-UNITARYPIPE	9	11	10	2	0.59
4	POP-UNITARYPIPE	9	11	10	2	0.59

Cuadro A.9: Esquemas de acción del dominio *Pipesworld* ordenados por valor de spliteabilidad.

A.9. Zenotravel

Este dominio modela un problema de transporte logístico de pasajeros a través de aeronaves comerciales que viajan a distintas ciudades. Cada viaje realizado por una aeronave consume una cierta cantidad combustible. La meta consiste en ubicar a cada pasajero en su ciudad de destino con las restricciones de combustibles dada para cada aeronave. Este dominio cuenta con:

- cantidad de esquemas de acción: 5
- cantidad de predicados: 5
- promedio tamaño de interfaz: 4.2
- máximo tamaño de interfaz: 6
- valor de spliteabilidad: 0.57

En el cuadro A.10 presentamos los esquemas de acción del dominio con su correspondiente tamaño de interfaz (Int), cantidad de precondiciones atómicas (Pre), cantidad de efectos atómicos (Eff), máxima interfaz entre las precondiciones y efectos atómicos (At), valor de spliteabilidad (Splty).

Zenotravel						
	Esquema	**Int**	**Pre**	**Eff**	**At**	**Splty**
1	ZOOM	6	4	4	2	0.57
2	FLY	5	3	4	2	0.54
3	REFUEL	4	3	2	2	0.48
4	BOARD	3	2	2	2	0.34
5	DEBARK	3	2	2	2	0.34

Cuadro A.10: Esquemas de acción del dominio *Zenotravel* ordenados por valor de spliteabilidad.

A.10. Scanalyzer

Este dominio modela un problema de invernaderos inteligentes, más conocidas en inglés como "green smart houses". Una casa verde consta de uno o más invernaderos junto con instalaciones para toma de imágenes para recopilar datos sobre las plantas que se cultivan, con el fin optimizar su rendimiento. La toma de imágenes deben funcionar de manera totalmente autónoma. Para este propósito, las plantas son transportadas entre los invernaderos y las instalaciones de imágenes por un sistema de cintas transportadoras. El dominio fue concebido para modelar el problema de controlar dichas cintas. Este dominio cuenta con:

- cantidad de esquemas de acción: 4
- cantidad de predicados: 7
- promedio tamaño de interfaz: 6.0
- máximo tamaño de interfaz: 8
- valor de spliteabilidad: 0.57

En el cuadro A.11 presentamos los esquemas de acción del dominio con su correspondiente tamaño de interfaz (Int), cantidad de precondiciones atómicas (Pre), cantidad de efectos atómicos (Eff), máxima interfaz entre las precondiciones y efectos atómicos (At), valor de spliteabilidad (Splty).

Scanalyzer						
	Esquema	**Int**	**Pre**	**Eff**	**At**	**Splty**
1	ANALYZE-4	8	5	9	2	0.57
2	ROTATE-4	8	5	8	2	0.57
3	ROTATE-2	4	3	4	2	0.45
4	ANALYZE-2	4	3	5	2	0.45

Cuadro A.11: Esquemas de acción del dominio *Scanalyzer* ordenados por valor de spliteabilidad.

A.11. Hiking

Este dominio modela una competencia de camping en parejas. Cada pareja deben llegar a su base de destino atravesando diferentes lugares permitidos de acampe. Cada pareja cuenta con una tienda para pasar la noche y continuar con su itinerario al día siguiente. Este dominio cuenta con:

- cantidad de esquemas de acción: 7
- cantidad de predicados: 9
- promedio tamaño de interfaz: 4.57
- máximo tamaño de interfaz: 6
- valor de spliteabilidad: 0.55

En el cuadro A.12 presentamos los esquemas de acción del dominio con su correspondiente tamaño de interfaz (Int), cantidad de precondiciones atómicas (Pre), cantidad de efectos atómicos (Eff), máxima interfaz entre las precondiciones y efectos atómicos (At), valor de spliteabilidad (Splty).

Hiking						
	Esquema	**Int**	**Pre**	**Eff**	**At**	**Splty**
1	DRIVE_TENT_PASSENGER	6	6	8	2	0.55
2	DRIVE_TENT	5	4	6	2	0.52
3	WALK_TOGETHER	6	8	6	3	0.52
4	DRIVE_PASSENGER	5	4	6	2	0.51
5	DRIVE	4	2	4	2	0.48
6	PUT_DOWN	3	3	2	1	0.41
7	PUT_UP	3	3	2	1	0.41

Cuadro A.12: Esquemas de acción del dominio *Hiking* ordenados por valor de spliteabilidad.

A.12. Cavediving

Este dominio consiste de un conjunto de buzos, cada uno de los cuales puede transportar cuatro tanques de aire. Los buzos deben ir a diferentes cuevas submarinas y tomar fotografías de las mismas. La cuevas son demasiado estrechas para que entre más de un buzo a la vez. El sistema de cuevas está representado por un gráfico acíclico no dirigido. Los buzos tienen un único punto de entrada. Nadar y fotografiar son acciones que consumen aire de los tanques. Los buzos deben salir de la cueva y descomprimirse al final. Por lo tanto, solo pueden visitar una cueva por viaje. Este dominio cuenta con:

- cantidad de esquemas de acción: 8
- cantidad de predicados: 17
- promedio tamaño de interfaz: 3.38
- máximo tamaño de interfaz: 5
- valor de spliteabilidad: 0.54

En el cuadro A.13 presentamos los esquemas de acción del dominio con su correspondiente tamaño de interfaz (Int), cantidad de precondiciones atómicas (Pre), cantidad de efectos atómicos (Eff), máxima interfaz entre las precondiciones y efectos atómicos (At), valor de spliteabilidad (Splty).

Cavediving						
	Esquema	**Int**	**Pre**	**Eff**	**At**	**Splty**
1	PREPARE-TANK	5	5	6	2	0.54
2	PICKUP-TANK	5	4	4	2	0.52
3	DROP-TANK	5	4	4	2	0.52
4	SWIM	4	4	3	2	0.48
5	PHOTOGRAPH	3	3	2	1	0.41
6	HIRE-DIVER	1	2	4	1	0.00
7	ENTER-WATER	2	2	2	1	0.00
8	DECOMPRESS	2	2	3	2	0.00

Cuadro A.13: Esquemas de acción del dominio *Cavediving* ordenados por valor de spliteabilidad.

Bibliografía

[Alcázar and Torralba, 2015] Vidal Alcázar and Álvaro Torralba. A reminder about the importance of computing and exploiting invariants in planning. In Ronen Brafman, Carmel Domshlak, Patrik Haslum, and Shlomo Zilberstein, editors, *Proceedings of the 25th International Conference on Automated Planning and Scheduling (ICAPS'15)*, pages 2–6. AAAI Press, 2015. [cited in page(s): 34]

[Alcázar *et al.*, 2014] Vidal Alcázar, Susana Fernández, and Daniel Borrajo. Analyzing the impact of partial states on duplicate detection and collision of frontiers. In Chien et al. [2014]. [cited in page(s): 34]

[Areces *et al.*, 2014] Carlos Areces, Facundo Bustos, Martín Dominguez, and Jörg Hoffmann. Optimizing planning domains by automatic action schema splitting. In Chien et al. [2014]. [cited in page(s): 131, 166]

[Bäckström and Nebel, 1993] Christer Bäckström and Bernhard Nebel. Complexity results for SAS^+ planning. In *Proceedings of the 13th International Joint Conference on Artificial Intelligence (IJCAI'93)*, pages 1430–1435, Chambery, France, August 1993. Morgan Kaufmann. [cited in page(s): 28]

[Bäckström and Nebel, 1995] Christer Bäckström and Bernhard Nebel. Complexity results for SAS^+ planning. *Computational Intelligence*, 11(4):625–655, 1995. [cited in page(s): 28, 133]

[Bäckström, 1992] Christer Bäckström. Equivalence and tractability results for SAS^+ planning. In Nebel et al. [1992], pages 126–137. [cited in page(s): 28]

[Blum and Furst, 1997] Avrim L. Blum and Merrick L. Furst. Fast planning through planning graph analysis. *Artificial Intelligence*, 90(1–2):279–298, 1997. [cited in page(s): 115]

[Boddy *et al.*, 2005] Mark Boddy, Jonathan Gohde, Tom Haigh, and Steven Harp. Course of action generation for cyber security using classical planning. In Susanne Biundo, Karen Myers, and Kanna Rajan, editors, *Proceedings of the 15th International Conference on Automated Planning and Scheduling (ICAPS-05)*, pages 12–21, Monterey, CA, USA, 2005. Morgan Kaufmann. [cited in page(s): 90]

[Bonet and Geffner, 1998] Blai Bonet and Héctor Geffner. HSP: Heuristic search planner. In *AIPS-98 Planning Competition*, Pittsburgh, PA, 1998. [cited in page(s): 4]

[Bonet and Geffner, 2001] Blai Bonet and Héctor Geffner. Planning as heuristic search. *Artificial Intelligence*, 129(1–2):5–33, 2001. [cited in page(s): 37]

[Bonet and Helmert, 2010] Blai Bonet and Malte Helmert. Strengthening landmark heuristics via hitting sets. In Helder Coelho, Rudi Studer, and Michael Wooldridge, editors, *Proceedings of the 19th European Conference on Artificial Intelligence (ECAI'10)*, pages 329–334, Lisbon, Portugal, August 2010. IOS Press. [cited in page(s): 38]

[Borrajo *et al.*, 2013] Daniel Borrajo, Simone Fratini, Subbarao Kambhampati, and Angelo Oddi, editors. *Proceedings of the 23rd International Conference on Automated Planning and Scheduling (ICAPS'13)*, Rome, Italy, 2013. AAAI Press. [cited in page(s): 185, 189]

[Botea *et al.*, 2005] Adi Botea, Markus Enzenberger, Martin Müller, and Jonathan Schaeffer. Macro-FF: Improving AI planning with automatically learned macro-operators. *Journal of Artificial Intelligence Research*, 24:581–621, 2005. [cited in page(s): 57]

[Bylander, 1994] Tom Bylander. The computational complexity of propositional STRIPS planning. *Artificial Intelligence*, 69(1–2):165–204, 1994. [cited in page(s): 4, 15]

[Cesta and Borrajo, 2001] A. Cesta and D. Borrajo, editors. *Proceedings of the 6th European Conference on Planning (ECP'01)*. Springer-Verlag, 2001. [cited in page(s): 184, 186, 189]

[Chien *et al.*, 2000] S. Chien, R. Kambhampati, and C. Knoblock, editors. *Proceedings of the 5th International Conference on Artificial Intelligence Planning Systems (AIPS'00)*, Breckenridge, CO, 2000. AAAI Press, Menlo Park. [cited in page(s): 185]

[Chien *et al.*, 2014] Steve Chien, Minh Do, Alan Fern, and Wheeler Ruml, editors. *Proceedings of the 24th International Conference on Automated Planning and Scheduling (ICAPS'14)*. AAAI Press, 2014. [cited in page(s): 183, 186, 189]

[Culberson and Schaeffer, 1998] Joseph C. Culberson and Jonathan Schaeffer. Pattern databases. *Computational Intelligence*, 14(3):318–334, 1998. [cited in page(s): 38]

[Edelkamp, 2001] Stefan Edelkamp. Planning with pattern databases. In Cesta and Borrajo [2001], pages 13–24. [cited in page(s): 38]

[Fox and Gomes, 2008] Dieter Fox and Carla Gomes, editors. *Proceedings of the 23rd National Conference of the American Association for Artificial Intelligence (AAAI'08)*, Chicago, Illinois, USA, July 2008. AAAI Press. [cited in page(s): 186, 189]

[Gazen and Knoblock, 1997] B. Cenk Gazen and Craig Knoblock. Combining the expressiveness of UCPOP with the efficiency of Graphplan. In Steel and Alami [1997], pages 221–233. [cited in page(s): 17, 57]

[Gerevini *et al.*, 2009] Alfonso Gerevini, Adele Howe, Amedeo Cesta, and Ioannis Refanidis, editors. *Proceedings of the 19th International Conference on Automated Planning and Scheduling (ICAPS'09)*. AAAI Press, 2009. [cited in page(s): 185, 186]

[Gnad *et al.*, 2019] Daniel Gnad, Álvaro Torralba, Martín Domínguez, Carlos Areces, and Facundo Bustos. Learning how to ground a plan – partial grounding in classical planning. In Pascal Van Hentenryck and Zhi-Hua Zhou, editors, *Proceedings of the 33nd AAAI Conference on Artificial Intelligence (AAAI'19)*. AAAI Press, January 2019. [cited in page(s): 49]

[Haslum and Geffner, 2000] Patrik Haslum and Hector Geffner. Admissible heuristics for optimal planning. In Chien et al. [2000], pages 140–149. [cited in page(s): 37]

[Haslum and Jonsson, 2000] Patrik Haslum and Peter Jonsson. Planning with reduced operator sets. In Chien et al. [2000], pages 150–158. [cited in page(s): 57]

[Haslum *et al.*, 2007] Patrik Haslum, Adi Botea, Malte Helmert, Blai Bonet, and Sven Koenig. Domain-independent construction of pattern database heuristics for cost-optimal planning. In Adele Howe and Robert C. Holte, editors, *Proceedings of the 22nd National Conference of the American Association for Artificial Intelligence (AAAI'07)*, pages 1007–1012, Vancouver, BC, Canada, July 2007. AAAI Press. [cited in page(s): 38]

[Haslum *et al.*, 2013] Patrik Haslum, Malte Helmert, and Anders Jonsson. Safe, strong, and tractable relevance analysis for planning. In Borrajo et al. [2013]. [cited in page(s): 132]

[Haslum, 2007] Patrik Haslum. Reducing accidental complexity in planning problems. In Manuela Veloso, editor, *Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI'07)*, pages 1898–1903, Hyderabad, India, January 2007. Morgan Kaufmann. [cited in page(s): 57]

[Haslum, 2009] Patrik Haslum. $h^m(P) = h^1(P^m)$: Alternative characterisations of the generalisation from $h^{\max}$ to h^m. In Gerevini et al. [2009], pages 354–357. [cited in page(s): 37]

[Haslum, 2011] Patrik Haslum. Computing genome edit distances using domain-independent planning. In *Proceedings of the 5th International Scheduling and Planning Applications woRKshop (SPARK)*, 2011. [cited in page(s): 90]

[Helmert and Domshlak, 2009] Malte Helmert and Carmel Domshlak. Landmarks, critical paths and abstractions: What's the difference anyway? In Gerevini et al. [2009], pages 162–169. [cited in page(s): 36, 37]

[Helmert and Mattmüller, 2008] Malte Helmert and Robert Mattmüller. Accuracy of admissible heuristic functions in selected planning domains. In Fox and Gomes [2008], pages 938–943. [cited in page(s): 37]

[Helmert *et al.*, 2014] Malte Helmert, Patrik Haslum, Jörg Hoffmann, and Raz Nissim. Merge & shrink abstraction: A method for generating lower bounds in factored state spaces. *Journal of the Association for Computing Machinery*, 61(3):16:1–16:63, 2014. [cited in page(s): 38]

[Helmert, 2006] Malte Helmert. The Fast Downward planning system. *Journal of Artificial Intelligence Research*, 26:191–246, 2006. [cited in page(s): 7, 28, 32, 48, 93, 113, 114, 133]

[Helmert, 2009] Malte Helmert. Concise finite-domain representations for PDDL planning tasks. *Artificial Intelligence*, 173:503–535, 2009. [cited in page(s): 6, 32, 49, 115, 131, 133]

[Heusner *et al.*, 2014] Manuel Heusner, Martin Wehrle, Florian Pommerening, and Malte Helmert. Under-approximation refinement for classical planning. In Chien et al. [2014]. [cited in page(s): 132]

[Hoffmann and Nebel, 2001a] Jörg Hoffmann and Bernhard Nebel. The FF planning system: Fast plan generation through heuristic search. *Journal of Artificial Intelligence Research*, 14:253–302, 2001. [cited in page(s): 37, 89, 132]

[Hoffmann and Nebel, 2001b] Jörg Hoffmann and Bernhard Nebel. RIFO revisited: Detecting relaxed irrelevance. In Cesta and Borrajo [2001], pages 325–336. [cited in page(s): 132]

[Hoffmann *et al.*, 2006] Jörg Hoffmann, Stefan Edelkamp, Sylvie Thíebaux, Roman Englert, Frederico Liporace, and Sebastian Trüg. Engineering benchmarks for planning: the domains used in the deterministic part of IPC-4. *Journal of Artificial Intelligence Research*, 26:453–541, 2006. [cited in page(s): 90]

[Hoffmann, 2005] Jörg Hoffmann. Where 'ignoring delete lists' works: Local search topology in planning benchmarks. *Journal of Artificial Intelligence Research*, 24:685–758, 2005. [cited in page(s): 37]

[Kambhampati *et al.*, 1997] Subbarao Kambhampati, Eric Parker, and Eric Lambrecht. Understanding and extending Graphplan. In Steel and Alami [1997], pages 260–272. [cited in page(s): 5]

[Karpas and Domshlak, 2009] Erez Karpas and Carmel Domshlak. Cost-optimal planning with landmarks. In Craig Boutilier, editor, *Proceedings of the 21st International Joint Conference on Artificial Intelligence (IJCAI'09)*, pages 1728–1733, Pasadena, California, USA, July 2009. Morgan Kaufmann. [cited in page(s): 37]

[Kautz and Selman, 1992] Henry A. Kautz and Bart Selman. Planning as satisfiability. In B. Neumann, editor, *Proceedings of the 10th European Conference on Artificial Intelligence (ECAI'92)*, pages 359–363, Vienna, Austria, August 1992. Wiley. [cited in page(s): 4]

[Kautz and Selman, 1998] Henry A. Kautz and Bart Selman. BLACKBOX: A new approach to the application of theorem proving to problem solving. In *Working notes of the AIPS'98 Workshop on Planning as Combinatorial Search*, Pittsburgh, PA, 1998. [cited in page(s): 4]

[Knoblock, 1994] Craig Knoblock. Automatically generating abstractions for planning. *Artificial Intelligence*, 68(2):243–302, 1994. [cited in page(s): 57]

[Koehler and Hoffmann, 2000] Jana Koehler and Jörg Hoffmann. On the instantiation of ADL operators involving arbitrary first-order formulas. In *Proceedings ECAI-00 Workshop on New Results in Planning, Scheduling and Design*, 2000. [cited in page(s): 17, 48]

[Kramer *et al.*, 2001] Stefan Kramer, Nada Lavrač, and Peter Flach. Propositionalization approaches to relational data mining. In *Relational data mining*, pages 262–291. Springer, 2001. [cited in page(s): 119]

[Lang and Toussaint, 2009] Tobias Lang and Marc Toussaint. Relevance grounding for planning in relational domains. In Wray L. Buntine, Marko Grobelnik, Dunja Mladenic, and John Shawe-Taylor, editors, *Proceedings of the Machine Learning and Knowledge Discovery in Databases, European Conference, ECML PKDD*, volume 5781 of *Lecture Notes in Computer Science*, pages 736–751. Springer, 2009. [cited in page(s): 114]

[Masoumi *et al.*, 2015] Arman Masoumi, Megan Antoniazzi, and Mikhail Soutchanski. Modeling organic chemistry and planning organic synthesis. volume 36 of *EPiC Series in Computing*, pages 176–195, 2015. [cited in page(s): 166]

[Matloob and Soutchanski, 2016] Rami Matloob and Mikhail Soutchanski. Exploring organic synthesis with state-ofthe-art planning techniques. In *Proceedings of Scheduling and Planning Applications woRKshop (SPARK)*, pages 52–61, 2016. [cited in page(s): 46]

[McDermott *et al.*, 1998] Drew McDermott, Malik Ghallab, Adele Howe, Craig Knoblock, Ashwin Ram, Manuela Veloso, Daniel Weld, and David Wilkins. *The PDDL Planning Domain Definition Language*. The AIPS-98 Planning Competition Comitee, 1998. [cited in page(s): 5, 51, 77]

[Miller and Freund, 1973] Irwin Miller and John Freund. *Probabilidad y estadística para ingenieros*. Reverté, 1973. [cited in page(s): 128]

[Miller *et al.*, 2018] Doug Miller, Ron Alford, Andy Applebaum, Henry Foster, Caleb Little, and Blake Strom. Automated adversary emulation: A case for planning and acting with unknowns. Technical report, The MITRE Corporation, 2018. [cited in page(s): 114]

[Nebel *et al.*, 1992] B. Nebel, W. Swartout, and C. Rich, editors. *Principles of Knowledge Representation and Reasoning: Proceedings of the 3rd International Conference (KR-92)*, Cambridge, MA, October 1992. Morgan Kaufmann. [cited in page(s): 183, 189]

[Nebel *et al.*, 1997] Bernhard Nebel, Yannis Dimopoulos, and Jana Koehler. Ignoring irrelevant facts and operators in plan generation. In Steel and Alami [1997], pages 338–350. [cited in page(s): 132]

[Nebel, 2000] Bernhard Nebel. On the compilability and expressive power of propositional planning formalisms. *Journal of Artificial Intelligence Research*, 12:271–315, 2000. [cited in page(s): 17, 57]

[Newton *et al.*, 2007] M. A. Hakim Newton, John Levine, Maria Fox, and Derek Long. Learning macro-actions for arbitrary planners and domains. In Mark Boddy, Maria Fox, and Sylvie Thiebaux, editors, *Proceedings of the 17th International Conference on Automated Planning and Scheduling (ICAPS'07)*, pages 256–263, Providence, Rhode Island, USA, 2007. Morgan Kaufmann. [cited in page(s): 57]

[Palacios and Geffner, 2009] Hector Palacios and Hector Geffner. Compiling uncertainty away in conformant planning problems with bounded width. *Journal of Artificial Intelligence Research*, 35:623–675, 2009. [cited in page(s): 57]

[Pednault, 1989] Edwin P.D. Pednault. ADL: Exploring the middle ground between STRIPS and the situation calculus. In R. Brachman, H. J. Levesque, and R. Reiter, editors, *Principles of Knowledge Representation and Reasoning: Proceedings of the 1st International Conference (KR-89)*, pages 324–331, Toronto, ON, May 1989. Morgan Kaufmann. [cited in page(s): 77]

[Pedregosa *et al.*, 2011] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay.

Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011. [cited in page(s): 135]

[Penberthy and Weld, 1992] J. Scott Penberthy and Daniel S. Weld. UCPOP: A sound, complete, partial order planner for ADL. In Nebel et al. [1992], pages 103–114. [cited in page(s): 5, 131]

[Porteous *et al.*, 2001] Julie Porteous, Laura Sebastia, and Jörg Hoffmann. On the extraction, ordering, and usage of landmarks in planning. In Cesta and Borrajo [2001], pages 37–48. [cited in page(s): 37]

[Richter and Westphal, 2010] Silvia Richter and Matthias Westphal. The LAMA planner: Guiding cost-based anytime planning with landmarks. *Journal of Artificial Intelligence Research*, 39:127–177, 2010. [cited in page(s): 96, 137]

[Richter *et al.*, 2008] Silvia Richter, Malte Helmert, and Matthias Westphal. Landmarks revisited. In Fox and Gomes [2008], pages 975–982. [cited in page(s): 37]

[Ridder and Fox, 2014] Bram Ridder and Maria Fox. Heuristic evaluation based on lifted relaxed planning graphs. In Chien et al. [2014], pages 244–252. [cited in page(s): 131]

[Röger *et al.*, 2018] Gabriele Röger, Silvan Sievers, and Michael Katz. Symmetry-based task reduction for relaxed reachability analysis. In *Proceedings of the 28th International Conference on Automated Planning and Scheduling (ICAPS'18)*, pages 208–217. AAAI Press, 2018. [cited in page(s): 131]

[Seipp and Helmert, 2013] Jendrik Seipp and Malte Helmert. Counterexample-guided Cartesian abstraction refinement. In Borrajo et al. [2013], pages 347–351. [cited in page(s): 38]

[Steel and Alami, 1997] S. Steel and R. Alami, editors. *Proceedings of the 4th European Conference on Planning (ECP'97)*. Springer-Verlag, 1997. [cited in page(s): 185, 187, 188]

[Tarjan, 1972] Robert E. Tarjan. Depth first search and linear graph algorithms. *SIAM Journal on Computing*, 1(2):146–160, 1972. [cited in page(s): 80]

[Torralba and Kissmann, 2015] Álvaro Torralba and Peter Kissmann. Focusing on what really matters: Irrelevance pruning in merge-and-shrink. In Levi Lelis and Roni Stern, editors, *Proceedings of the 8th Annual Symposium on Combinatorial Search (SOCS'15)*, pages 122–130. AAAI Press, 2015. [cited in page(s): 132]

[Torralba *et al.*, 2017] Álvaro Torralba, Vidal Alcázar, Peter Kissmann, and Stefan Edelkamp. Efficient symbolic search for cost-optimal planning. *Artificial Intelligence*, 242:52–79, 2017. [cited in page(s): 119]

[Toyer *et al.*, 2018] Sam Toyer, Felipe Trevizan, Sylvie Thiebaux, and Lexing Xie. Action schema networks: Generalised policies with deep learning. In Sheila McIlraith and Kilian Weinberger, editors, *Proceedings of the 32nd AAAI Conference on Artificial Intelligence (AAAI'18)*. AAAI Press, February 2018. [cited in page(s): 132]

[Younes and Simmons, 2003] Håkan L. S. Younes and Reid G. Simmons. VHPOP: versatile heuristic partial order planner. *Journal of Artificial Intelligence Research*, 20:405–430, 2003. [cited in page(s): 5]