Generación automática de conocimiento de control en un planificador híbrido HTN-POP

Susana Fernández, Ricardo Aler y Daniel Borrajo

Departamento de Informática, Universidad Carlos III of Madrid, 28911 Leganés
email:{sfarregu@inf aler@inf dborrajo@ia}.uc3m.es

Resumen Este trabajo presenta un sistema de aprendizaje automático, HHAMLET, para un planificador híbrido, HYBIS, que mezcla planificación jerárquica HTN (*Hierachical Task Network*) y planificación de orden parcial (POP). HHAMLET está basado en HAMLET, un sistema multi-estrategia deductivo-inductivo, que aprende reglas de control para el planificador no lineal de orden total PRODIGY4.0. Los dominios en HYBIS representan plantas indrustriales y se definen de manera jerárquica. Resuelve problemas de manufacturación, descomponiéndolos en problemas de mayor a menor nivel de abstracción. En cada nivel se genera un plan utilizando POP. Una vez encontrado el plan en un nivel, cada acción se transforma en un problema del nivel inferior según un método de expansión. El aprendizaje puede tener lugar durante la descomposición jeráquica, para aprender el método de expansión más adecuado o bien, durante la generación de los planes de orden parcial.

1. Introducción

En este trabajo se presenta HHAMLET, un sistema que aprende conocimiento de control para mejorar la eficiencia de un planificador HTN-POP, diseñado para la resolución de problemas del mundo real, llamado HYBIS [5]. HYBIS mezcla técnicas de planificación HTN [7] y Planificación de Orden Parcial (POP) [17], para generar secuencias de control en un proceso de manufacturación. El dominio es el modelo de conocimiento de la planta industrial donde se elaboran los productos. Se compone de un conjunto de agentes y de un número de axiomas que describen hechos que siempre se verifican. Cada agente representa un dispositivo industrial, junto con las operaciones o acciones que es capaz de realizar. La representación de sus operadores está basada en autómatas finitos, están en un estado y por efecto de una acción cambian a otro diferente. Los agentes se describen jerárquicamente según las diferentes partes de las que esté compuesto, que a su vez pueden ser otros agentes. Los propios agentes, tiene información de como hacer la descomposición jerárquica de las tareas y siempre tienen definido un método por omisión. Sin embargo, HYBIS permite definir otros métodos de descomposición, específicos del dominio, para casos en que el método por omisión no sea suficiente. Un problema consiste en obtener la secuencia de control que pase de unos productos iniciales base a otros elaborados. Como todo problema de planificación, se representa por un estado inicial y unas metas. El estado

inicial es el conjunto de literales que describen tanto el sistema de manufacturación como los productos base y las metas, representan el conjunto de literales que describen las transformaciones necesarias para obtener el producto manufacturado. Como ocurre en todos los planificadores independientes del dominio, HYBIS no encuentra el mejor plan rápidamente porque gasta mucho tiempo estudiando alternativas no válidas antes de llegar a la solución. Para evitar esto, proponemos adquirir conocimiento automáticamente para, posteriormente, guiar el proceso de planificación. Este conocimiento se adquiere por la experiencia de resolver problemas anteriores al generar una explicación de los pasos seguidos por el propio proceso de planificación. En planificación se han desarrollado con éxito varias aproximaciones que guían el proceso de búsqueda por medio de la adquisición de conocimiento de control, bien aprendiéndolo automáticamente [1,4,8,10,14,16], o bien añadiéndolo a mano directamente por un humano [3]. Quizá, el esquema más básico para adquirir este conocimiento es por medio de técnicas de aprendizaje deductivo que generan reglas de control a partir de un único o varios problemas resueltos. Este es el caso de técnicas puras EBL [12,14], y técnicas basadas en ella [1]. Las reglas de control obtenidas son usadas posteriormente durante la ejecución de nuevos problemas, para podar el árbol de búsqueda y reducir el espacio de búsqueda. Estas reglas permiten mejorar tanto la eficiencia de la búsqueda como, en algunos casos, mejorar la calidad de los planes generados. Las técnicas de aprendizaje automático también se han extendido a la planificación jerárquica. Ejemplo de ello son los sistemas KNOMIC [15] y CAMEL [11] y los trabajos de Garland, Ryall y Rich [9] y los de Lotem y Dana S. Nau [13].

El trabajo está organizado en cinco secciones. La sección 2 es una descripción del planificador y de los dominios de manufacturación a los que se aplica. La sección 3 describe el proceso de aprendizaje y las adaptaciones necesarias para tratar estos dominios. La sección 4 muestra algunos resultados experimentales obtenidos sobre diferentes dominios. Por último, la sección 5 son las conclusiones y los trabajos futuros.

2. El planificador. HYBIS

El diseño correcto y completo de un programa de control industrial es muy complejo, incluso para un humano. Tradicionalmente los ingenieros de control han usado diferentes metodologías, estándares y herramientas para llevar a cabo esta tarea. El estándar ISA-SP88 [2] constituye una de estas metodologías usada para hacer un diseño jerárquico de programas de control industrial. El planificador HYBIS mezcla técnicas de HTN y POCL para aproximar la resolución de problemas de planificación a la manera en que los ingenieros diseñan los programas de control en los sistemas de manufacturación en dos sentidos: representa una planta industrial como una composición jerárquica de dispositivos a distintos niveles de abstracción que aceptan el estándar SP88 y genera programas de control automáticamente a distintos niveles de detalle.

Los dominios para el planificador son representados por una jerarquía de agentes donde la raíz (un agente ficticio) representa toda la planta industrial, los nodos hojas son los **agentes primitivos** que representan los dispositivos de la planta y los nodos intermedios son los **agentes agregados**. La estructura y comportamiento de los agentes agregados viene determinada por el conjunto de agentes de un nivel menor de abstracción del que estén compuestos. Cada agente agregado tiene conocimiento de las diferentes alternativas que haya para llevar a cabo las acciones del siguiente nivel. Esto es equivalente a los diferentes métodos usados en HTN para descomponer cada operador. Cada agente tiene definidas unas acciones que puede realizar. Las acciones de los agentes agregados pueden tener definida una propiedad llamada **expansión** que especifica las diferentes maneras posibles de transformar una acción en un problema de otro nivel. Cada método de expansión se representa por un conjunto de literales, que pueden estar ordenados, representando un problema a resolver por agentes del siguiente nivel de abstracción. Todos los agentes agregados tienen siempre una propiedad llamada **interfaz** que constituye el método de expansión por omisión. Si una acción agregada no tiene definido un método de expansión, la función de expandir una acción devuelve una lista de literales resultado de aplicar la interfaz de su agente a los literales de los efectos de la acción. El método por omisión es independiente del dominio pues sólo es una traducción de los efectos de una acción agregada a literales (metas) del siguiente nivel que deberán resolverse por subacciones de la acción compuesta. Una misma acción podría descomponerse de formas distintas (en el sentido de las subacciones del otro nivel) siguiendo el método por omisión, según el contexto del problema.

2.1. Algoritmo de Planificación

El proceso de planificación es un algoritmo generativo y regresivo a distintos niveles de detalle. Cada plan en un nivel de abstracción es refinado en otro plan de menor nivel hasta que no quede ninguna acción primitiva en el plan de menor nivel de abstracción (y mayor nivel en la jerarquía). En cada nivel, el plan es generado por MACHINE [6], un Planificador de Orden Parcial. La entrada al planificador completo HYBIS es la descripción del dominio (jerarquía de agentes), el estado inicial que es el conjunto de literales representando las condiciones iniciales de los estados de todos los agentes del dominio y un conjunto de objetivos al mayor nivel de abstracción. El procedimiento es el siguiente:

- Primero, por medio de un proceso generativo POP se obtiene la secuencia de acciones de control que deben ser resueltas por los agentes del mayor nivel de abstracción
- Segundo, si la secuencia encontrada está compuesta sólo por acciones primitivas entonces el problema está resuelto. Si no, la secuencia es jerárquicamente refinada, es decir, el algoritmo expande cada actividad agregada según la interfaz del agente o los métodos de expansión que tenga definido, para obtener un nuevo problema de un nivel menor
- Tercero, el algoritmo se repite recursivamente para resolver el problema con los agentes del siguiente nivel

Por tanto, el plan final obtenido por este algoritmo es la secuencia de control a diferentes niveles de modularidad. Para más detalles sobre el algoritmo de planificación ver [5].

3. El proceso de aprendizaje

Para adquirir conocimiento de control de este planificador HTN-POCL seguimos un algoritmo con tres pasos:

1. El planificador se ejecuta sobre un problema y se etiquetan todos los nodos del árbol de búsqueda de forma que se puedan identificar los puntos de decisión de éxito, los nodos que conducen a la solución
2. Se seleccionan algunos de los puntos de decisión de éxito y se generan reglas de control a partir de ellos, de forma que cuando se vuelva a ejecutar el planificador otra vez con las reglas, vaya directamente a la solución
3. Las constantes y variables en las reglas de control se generalizan para que puedan ser usadas en otros problemas

Como en HAMLET, el aprendizaje en HHAMLET comienza después de que el planificador haya encontrado una solución. Dado que HYBIS devuelve el árbol de búsqueda completo en todos los niveles de la jerarquía, HHAMLET puede generar reglas de control para varios (o todos los) niveles de abstracción.

3.1. Etiquetado del árbol

Cada nodo del árbol de búsqueda tiene un campo etiqueta que puede tomar los siguientes valores:

- *success*, el nodo pertenece al camino solución del problema
- *failure*, pertenece a un camino de fallo
- *abandoned*, el planificador comienza a expandir este nodo pero lo abandona debido a la heurística que le recomienda seguir por otro camino
- *unknown*, el planificador no ha expandido el nodo

Durante el proceso de planificación, cada vez que se genera un nodo se etiqueta como *unknown*. Si se detecta un fallo, el nodo afectado se etiqueta como *failure*. Cuando se llega a una solución se van etiquetando todos los antecesores del nodo como *success*. Cuando empieza el proceso de aprendizaje, lo primero que se hace es etiquetar el árbol mediante un algoritmo recursivo ascendente, empezando por los nodos más profundos en el árbol y subiendo hasta el nodo raíz:

- Si un nodo tiene al menos un sucesor que sea de éxito, se etiqueta como *success* (esto, se hace justo al encontrar la solución)
- Si todos sus sucesores son de fallo él también se etiqueta como *failure*

- Si está etiquetado como *unknown*, pero tiene al menos un sucesor, se cambia su etiqueta a *abandoned*
- El resto se consideran *unknown*

Una vez que se tiene el árbol etiquetado existen dos puntos de decisión; es decir, nodos que son candidatos para que se aprenda una regla de control:

- *Failure-Success*: son nodos que tienen, al menos, dos ramas una con un nodo de éxito y otra de fallo
- *Abandoned-Success*: igual que antes pero en vez de fallo se trata de un nodo abandonado

Cuando encuentra uno de estos dos puntos de decisión, se genera una regla de control según se explica a continuación.

3.2. Generación de reglas de control

En los nodos de decisión donde se ha decidido aprender, se genera una regla de control para que el planificador pueda seleccionar la opción de éxito la próxima vez. Se pueden considerar tres tipos de reglas: de selección, de rechazo y de preferencia. Las de selección obligan al planificador a ejecutar una acción, las de rechazo le obligan a descartar un nodo y las de preferencia, exploran esa opción primero pero sin descartar las otras. De momento nos centramos en las de selección.

En un planificador HTN-POCL hay dos tipos diferentes de nodos donde se pueden aprender reglas:

1. Puntos HTN: de qué manera se puede hacer un refinamiento, es decir, qué método de expansión de los posibles es mejor elegir; y
2. Puntos POCL
   a) Para elegir un operador que ya está en el plan o añadir uno nuevo del dominio cuando se intenta hacer cierta una meta pendiente
   b) Para decidir en ambos casos qué operador seleccionar
   c) Para resolver una amenaza, si usar *promote* o *demote*

En este trabajo estudiamos el problema de la selección de un operador que combina los tipos 2a y 2b. Concretamente aprendemos dos tipos de reglas `SELECT Operator-new` y `SELECT Operator-plan` según lo que el planificador hiciese cuando estaba resolviendo una submeta pendiente; seleccionar un nuevo operador del dominio o reutilizar una acción que ya estuviese en el plan parcial.

Cada tipo de regla de control tienen una plantilla para describir sus precondiciones. Aunque compartan muchas de las características hay otras particulares para cada tipo de regla. Ejemplo de características comunes que llegan a ser metapredicados, (son metapredicados porque sus argumentos son predicados), del lenguaje de control son: htn-level, true-in-state, current-goal, some-candidate-goals. Ejemplos de metapredicados locales para cada tipo de regla son: operator-in-plan y operator-not-in-plan. Se describen más adelante.

Las reglas constan de tres partes: el nombre de la regla, la parte condicional o requisitos que debe cumplir, y la parte del consecuente o decisión a tomar. La parte condicional va precedida por *IF* y, a su vez, se compone de:

- El metapredicado `Htn-level`, su argumento se extrae del nivel en el que está el planificador jerárquico. No tiene sentido aplicar reglas aprendidas para un nivel en otro
- El metapredicado `Current-goal`, su argumento se extrae de la meta que está tratando de alcanzar el planificador
- El metapredicado `Some-candidate-goals`, su argumento son las otras submetas que deberán resolverse en ese nodo
- El metapredicado `Operator-not-in-plan`, para comprobar que, en las reglas de tipo `Operator-new`, la acción que se pasa como argumento no esté ya incluida en el plan parcial; o el metapredicado `Operator-in-plan`, para asegurarse de que, en las reglas de tipo `Operator-plan`, realmente se encuentre la acción pasada como argumento
- Por último hay un metapredicado `True-in-state` por cada literal que es cierto en el estado inicial. Para hacer más eficientes las reglas de control y disminuir los metapredicados `True-in-state`, se hace una regresión de metas. Es decir, sólo se consideran los literales del estado inicial que sean requeridos, directa o indirectamente, por las precondiciones del operador que añade la regla. Esta regresión de metas se realiza a través del conjunto de enlaces causales.

Cuando se generan las reglas, los argumentos de los metapredicados son o bien constantes, porque representan objetos particulares del dominio o bien variables de planificación que cambian en cada ejecución. Para evitar que las reglas de control dependan de los nombres particulares usados cuando se aprendió es preciso generalizarlas convirtiéndolas en variables dentro de las reglas, proceso denominado parametrización. Cada variable se puede emparejar sólo con objetos del mismo tipo. Por eso cada variable se nombra con un prefijo compuesto por el tipo y los caracteres % %. Por ejemplo, <still % %still1-agg> representa una variable que es del tipo `still`. No todas las constantes se pueden parametrizar; en algunos casos no tiene sentido. Por ejemplo, si tenemos el literal `(state trans-3 off)`, `trans-3` es un buen candidato para ser parametrizado, pero `off` no, ya que el significado de que el transporte está apagado se perdería. Por tanto, actualmente, no se generaliza el segundo argumento de los predicados `state`, por la semántica particular asosicada a ellos que tienen para este planificador.

La figura 1 muestra una regla de control generada con el proceso anterior. Es una regla de `Operator-new` que selecciona una nueva acción que no está en el plan parcial `filtrate(<filter>,<result>)`, cuando el planificador decide trabajar en la meta `contains(<i3>,<?src330>)`, teniendo pendiente de resolver alguna de las submetas que aparecen en el metapredicado `Some-candidate-goals` y son ciertos en el estado inicial los literales que aparecen en los metapredicados `True-in-state`.

```
(control-rule regla-1
 (if (and (htn-level 1)
          (current-goal (contains <i3> <?src330>))
          (some-candidate-goals ((state <line-3prod%%trans-3> off)
                                 (state <still%%still1-agg> off)
                                 (state <still%%still1-agg> ready)))
          (operator-not-in-plan (filtrate
                                 <filter-agg%%filteragg-agg>
                                 <?result>))
          (true-in-state (state <line-3prod%%trans-3> off))
          (true-in-state (state <filter-agg%%filteragg-agg> off))))
 (then select operator-new
             (filtrate <filter-agg%%filteragg-agg> <?result>)))
```

Figura1. Ejemplo regla de control.

4. Experimentación y resultados HYBIS

Hasta el momento, hemos hecho un estudio de la viabilidad de este método de aprendizaje, comprobando que los metapredicados utilizados en la parte izquierda de las reglas de control (las precondiciones) son los adecuados. Si las precondiciones de la reglas no son lo suficientemente significativas y completas para diferenciar todas las situaciones durante el proceso de planificación, se pueden disparar reglas de control en momentos erróneos, haciendo que el planificador seleccione un operador incorrecto. Al ser reglas de selección, el resto de alternativas se descartan, con lo cual se pierde la posibilidad de explorar el nodo con la acción adecuada y el sistema nunca llega a la solución. Dada las caraterísticas de los dominios tratados, en que hay muchos agentes de un determinado tipo y todos con las mismas acciones, se hace especialmente relevante esta cuestión. La manera más inmediata de comprobar la validez de este método de aprendizaje es haciendo pruebas con un mismo problema y un dominio. Para ello, generamos las reglas de control en todos los niveles de abstracción, ejecutando el planificador con un problema y dominio. Posteriormente, volvemos a ejecutar el proceso de planificación, con ese mismo problema, pero utilizando las reglas de control obtenidas anteriormente. Comparamos las dos ejecuciones, en cuanto al número de nodos explorados por el planificador y el tiempo que tarda en encontrar la solución, observando el ahorro producido al utilizar las reglas.

Esta validación podría sorprender, pero el concepto de dominio en HYBIS difiere del de otros planificadores. Para HYBIS un dominio es una planta industrial, diseñada específicamente para resolver un único problema, es decir, para generar un producto manufacturado concreto a partir de unas materias primas iniciales. Es por ello que se valida en un único problema. La pregunta entonces sería para qué es neceario un proceso de aprendizaje. Lo más usual es que se encuentre un único plan que realice esta transformación y el planificador no necesite volver a ejecutarse. Sin embargo, puede ocurrir que en una misma planta

industrial se requiera hacer una modificación del diseño, con lo que se tendría un nuevo dominio y problemas muy parecedidos al anterior. Para estos habría que volver a encontrar el plan adecuado, con todas las dificultades que ello conlleva. Esto es especialmente cierto hoy en día donde la tendencia es a fabricar bajo demanda, de forma flexible, permitiendo producir diferentes productos en la misma planta de forma rápida. Aunque la modificación sea mínima, el proceso de planificación tiene que empezar desde cero otra vez. Si se tienen reglas de control aprendidas en la primera ejecución y estas son correctas, este segundo problema de planificación se resolverá con mayor eficiencia.

Hemos hecho varios experimentos utilizando diferentes dominios con características distintas. En la tabla 1 se muestran las características de los dominios y problemas utilizados, como son el número de agentes definidos (Agentes), el número de niveles de abstracción (Niveles), el número de operadores o acciones (Operadores) que hay entre todos los agentes, el número de literales del estado inicial (Inicial) y el número de metas que tiene el problema resuelto (Metas).

Cuadro1. Características de los dominios

Dominio	Agentes	Niveles	Operadores	Inicial	Metas
ITOPS03	42	3	92	63	1
BC-2	19	2	44	27	5
PAPILLA	26	2	61	46	3
MANIPULADOR	15	3	46	26	1
PLANTA-3	10	2	20	16	2

En la tabla 2 se muestran los resultados al resolver un problema sin utilizar reglas y al resolver ese mismo problema usando las reglas aprendidas en una ejecución previa (del mismo problema). Se representa: el número de nodos generados por el proceso de planificación hasta encontrar la solución (Nodos), los segundos tardados hasta llegar a la solución (Tiempo), el porcentaje de ahorro en tiempo al utilizar las reglas (Ahorro) y el número total de reglas utilizadas de todos los niveles de abstracción (Reglas).

En todos se observa que disminuye tanto el número de nodos generados como el tiempo que tarda en llegar a la solución y aunque las diferencias no sean muy grandes, debido a la naturaleza de los problemas, hace pensar que cuando se utilice este esquema en problemas parecidos más complicados las reglas de control podrán ahorrar mucho tiempo de ejecución.

5. Conclusiones y trabajos futuros

En los planificadores HTN los planes se construyen en distintos niveles de jerarquía, empezando por el nivel más alto de abstracción y refinando poco a poco

Cuadro2. Ejecución planificador con y sin reglas

	Sin reglas		Con reglas			
Dominio	Nodos	Tiempo(s)	Nodos	Tiempo(s)	Ahorro %	Reglas
ITOPS03	898	244	639	212	13	59
BC-2	637	60	319	34	43	36
PAPILLA	583	62	326	40	35	40
MANIPULADOR	235	21	185	15	29	33
PLANTA-3	198	9	125	4	55	14

hasta niveles más específicos. Sin embargo, aunque el uso de la jerarquía simplifica la complejidad computacional, el proceso puede seguir siendo ineficiente. En un planificador híbrido como Hybis se puede incrementar la eficiencia tanto en los puntos de decisión propios de los HTN como en los de POP. Técnicas de aprendizaje automático se han usado en otros planificadores para mejorar el proceso de búsqueda aprendiendo del propio proceso de ejecución. En este trabajo hemos discutido varios aspectos de aprendizaje automático aplicado a este tipo de planificadores y hemos extendido algunas de las ideas de aprendizaje automático para poder aplicarlas a un planificador híbrido HTN-POP. Hemos generado reglas de control en un dominio resolviendo un problema y luego las hemos utilizado en ese mismo dominio y problema.

En un futuro pretendemos utilizar las reglas aprendidas en un dominio, en dominios similares. Por ejemplo, plantas industriales que tengan más o menos agentes del mismo tipo o dominios que compartan alguna parte de la jerarquía. También se pueden usar para resolver distintos problemas en un mismo dominio. Quisiéramos aprender reglas de todos los puntos de decisión posibles, incluidos los HTN. Pensamos extender el esquema de aprendizaje para que las reglas de control sean especializadas, generalizadas y combinadas, de manera inductiva resolviendo más problemas. Hybis es un planificador basado en agentes en donde unos agentes se componen de otros. Capturar esta información seria útil para añadir un valor semántico mayor a las reglas. También existe otra información sobre la manera en que se distribuyen los agentes en el dominio y sus conexiones con otros agentes que sería interesante capturar. Por último Hybis se ha extendido para poder generar planes condicionales lo cual ofrece nuevas oportunidades de aprendizaje.

Agradecimientos

Este trabajo ha sido parcialmente financiado por el MCyT a través de los proyecto TIC2001-4936-E y TAP1999-0535-C02-02. Los autores también quieren agradecer a Luis Castillo y Juan Fernández de la Universidad de Granada, por la ayuda ofrecida para el uso de Hybis.

Referencias

1. Ricardo Aler, Daniel Borrajo, and Pedro Isasi. Using genetic programming to learn and improve control knowledge. *Artificial Intelligence*, 2002.
2. ANSI/ISA. *Batch Control Part I, Models & Terminology (S88.01)*, 1995.
3. Fahiem Bacchus and Froduald Kabanza. Using temporal logics to express search control knowledge for planning. *Artificial Intelligence*, 116:123–191, 2000.
4. Daniel Borrajo and Manuela Veloso. Lazy incremental learning of control knowledge for efficiently obtaining quality plans. *AI Review Journal. Special Issue on Lazy Learning*, 11(1-5):371–405, February 1997.
5. Luis Castillo, Juan Fernández-Olivares, and Antonio González. A hybrid hierarchical/operator-based planning approach for the design of control programs. In *ECAI Workshop on Planning and configuration: New results in planning, scheduling and design*, 2000.
6. Luis Castillo, Juan Fernández-Olivares, and Antonio González. Mixing expressiveness and efficiency in a manufacturing planner. *Journal of Experimental and Theoretical Artificial Intelligence*, 13:141–162, 2001.
7. Ken Currie and Austin Tate. O-Plan: the open planning architecture. *Artificial Intelligence*, 52(1):49–86, 1991.
8. Tara A. Estlin and Raymond J. Mooney. Learning to improve both efficiency and quality of planning. In Martha Pollack, editor, *Proceedings of the 15th International Joint Conference on Artificial Intelligence (IJCAI-97)*, pages 1227–1232. Morgan Kaufmann, 1997.
9. A. Garland, K. Ryall, and C. Rich. Learning hierarchical task models by defining and refining examples. In *In First International Conference on Knowledge Capture*, 2001.
10. Yi-Cheng Huang, Bart Selman, and Henry Kautz. Learning declarative control rules for constraint-based planning. In Pat Langley, editor, *Proceedings of the Seventeenth International Conference on Machine Learning, ICML'00*, Stanford, CA (USA), June-July 2000.
11. Okhtay Ilghami, Dana S. Nau, Héctor Muñoz-Avila, and David W. Aha. Camel: Learning method preconditions for HTN planning. In *Proceedings of AIPS02*, 2002.
12. Subbarao Kambhampati. Improving Graphplan's search with EBL & DDB techniques. In Thomas Dean, editor, *Proceedings of the IJCAI'99*, pages 982–987, Stockholm, Sweden, July-August 1999. Morgan Kaufmann Publishers.
13. Amnon Lotem and Dana S. Nau. New advances in GraphHTN: Identifying independent subproblems in large HTN domains. In *Artificial Intelligence Planning Systems*, pages 206–215, 2000.
14. Steven Minton. *Learning Effective Search Control Knowledge: An Explanation-Based Approach*. PhD thesis, Computer Science Department, Carnegie Mellon University, 1988. Available as technical report CMU-CS-88-133.
15. Michael van Lent and Joha Laird. Learning hierarchical performance knowledge by observation. In *Proceedings of the 16th International Conference on Machine Learning*, pages 229–238, San Francisco, CA, 1999. Morgan Kaufmann.
16. Manuela Veloso. *Planning and Learning by Analogical Reasoning*. Springer Verlag, December 1994.
17. Daniel S. Weld. An introduction to least commitment planning. *AI Magazine*, 15(4):27–61, 1994.