Planeación

Eduardo Morales, Enrique Sucar

INAOE

Contenido

Planeación

Planeación involucra algunas de las principales áreas de IA:

- Búsqueda y toma de decisiones
- Representación de conocimiento
- Aprendizaje

Planeación

Tiene sus propios problemas:

- Representación y razonamiento temporal, causal y de intenciones
- Incertidumbre en la ejecución de un plan y consider al "mundo real"
- Percepciones y creencias del "mundo real"
- Agentes múltiples que cooperan/interfieren
- Restricciones físicas o de otro tipo sobre posibles soluciones
- Necesita justificar sus soluciones
- Extracción de conocimiento

Aplicaciones

Control de robots	STRIPS
Generación de programas	HACKER
Supervisión de aprendiz de ingeniero mecánico	NOAH
Construcción de casas	NONLIN
Genética molecular	MOLGEN
Viajes	OPM
programación (*scheduling*) para producción de turbinas	ISIS-II
Misión aviones de carga	SIPE, AIRPLAN
Logística naval	NONLIN
Secuenciación del Voyager	DEVISER
Conversaciones	KAMP
etc.	

Planeación

Planeación vs. Acción:

- Con simulación del mundo se puede hacer *backtracking*
- Calcular los pasos de un procedimiento de solución de problemas antes de ejecutarlos $=>$ plan = representación de secuencia de acciones

Planeación vs. Solución de problemas:

- "Un plan es un proceso jerárquico en un organismo que puede controlar el orden en que una secuencia de operaciones se deben de realizar" [Miller et al. 60]
- Operacionalmente la planeación trata de reducir la cantidad de trabajo para tratar de encontrar o producir un resultado correcto

Planeación

Implementaciones específicas vs generales:

- La interacción entre los sub-problemas depende mucho del dominio
- Técnicas específicas del dominio no son generalizables
- Técnicas generales no garantizan terminar a menos que se les impongan restricciones

Planeación

- La planeación produce, o controla la producción de, procedimientos, recetas, secuencias de operaciones o acciones, para conseguir el efecto deseado.
- "La planeación automática es un programa de computo que involucra la representación del mundo, representación de acciones y sus efectos en el mundo, razonamiento acerca de los efectos de secuencias de tales acciones, razonamiento acerca de las interacciones de las acciones que ocurren concurrentemente, y control de la búsqueda para que se puedan encontrar con eficiencia razonable los planes"

Planeación

Raíces: *Physical symbol systems hypothesis* [Newell, Simon, 76]
Modelos:

1. Búsqueda de estado-espacio
2. Descomposición de problemas

Búsqueda de Estado-Espacio

- Define un espacio de estados (espacio con todas las posibles soluciones potenciales) implicita/explicitamente enumerado
- Especifica los estados iniciales o situaciones de donde empezar
- Especifica los estados finales (metas) o aquellos reconocidos como soluciones
- Especifica las reglas que definen acciones u operaciones disponibles para moverse o ir de un estado a otro dentro del espacio del problema

Búsqueda de Estado-Espacio

- En este contexto, el proceso de solución de problemas trata de encontrar una secuencia de operaciones que transformen el estado inicial a uno final
- Con suficiente conocimiento, el solucionador puede saber qué hacer directamente
- En la práctica, todos involucran algo de búsqueda sistemática
- Se usa el conjunto de reglas junto con una *estrategia de control adecuada* para moverse en el espacio de búsqueda hasta encontrar un camino o solución del estado inicial al final

Búsqueda de Estado-Espacio

- En las estrategias de búsqueda, en general se necesitan *heurísticas*
- Las de proposito general se llaman “debiles” (*weak*)
  - genera y prueba
  - *breadth-first*
  - *best-first*
  - reducción de problemas
  - satisfacción de restricciones
  - *means-ends analysis*

Descomposición de Problemas

Los solucionadores de problemas, incluyendo planeación, involucran la combinación de:

- Una o más estrategias de solución básicas
- Uno o más mecanismos de representación de conocimiento
- Método de descomposición de problemas

Descomposición de Problemas

- Descripción de la meta del problema
- Descripción del estado inicial del problema
- Conjunto de descripciones de metas primitivas cuyas soluciones son inmediatas
- Conjunto de operadores para transformar metas a submetas (los operadores constituyen las transformaciones en que la descripción inicial del problema puede ser transformada a un conjunto de submetas cuyas soluciones son inmediatas)

Descomposición de Problemas

- En este contexto, la búsqueda de reducción de problemas es una grafo AND/OR, mientras que la búsqueda en el espacio de estados es una grafo OR
- En un grafo AND/OR:
  - Los nodos AND son submetas (las cuales se deben de cumplir todas para lograr la meta)
  - Los nodos OR representan descomposiciones alternas en diferentes submetas
- En un grafo OR: Los nodos OR son divisiones del camino en estados alternativos

Descomposición de Problemas

- En planeación, reducir una tarea implica encontrar una colección de subtareas tales que la tarea se pueda lograr al hacer todas las subtareas
- La colección de esas subtareas entonces representa un plan.
- El escoger conjuntos relevantes de subtareas (*OR choices*) es un problema por si mismo
- El planeador debe de poder considerar varias alternativas (e.g., búsqueda sistemática con *backtracking*)

Descomposición de Problemas

- El resolver un subproblema puede hacer inaplicable el operador requerido para resolver otro subproblema!!
- Una de las motivaciones de planeación es la de introducir herramientas o métodos para ayudar a los métodos débiles para reducir la búsqueda lo más posible.
- Los métodos débiles se dedican al control de la expansión exponencial, mientras que los métodos basados más en conocimiento o métodos fuertes, se dedican a su prevensión
- Planeacion queda entre “saber exactamente qué hacer” y “búsqueda ciega”

El *Frame Problem*

- Surge al tratar de formalizar interacciones con un mundo complejo
- Simplemente se refiere a cómo especificar los efectos de las acciones de una manera económica
- Concierne la dificultad de mantener información de las consecuencias de la realización de una acción en, o más generalmente en hacer las alteraciones a, la representación del mundo [Hayes]
- En escencia lo que dice es que hacer planeación en dominios reales requiere un modelado nada trivial

Los *Frame Axioms*

- Listan todo lo que se cree verdadero en una situación dada (no cambian por acciones), aunque las acciones afectaban sólo unas cuantas cosas, sistemas como el GPS "cargaban" con todos
- Mientras que en el GPS cada acción podía afectar la representación del mundo, en STRIPS cada acción se supone *no afectar* ningúna relación, a menos que se estipule
- No se puede tener cambios continuos (vs discretos)

El Mundo de los Bloques

- Es el *E. Coli* de los planificadores
- Posiblemente el problema más simple que no puede ser resuelto de manera óptima por planeadores lineales. E.g.:

Edo. Inicial	Edo. Final
(on C A) (cleartop B) (cleartop C)	(on A B) & (on B C)

El Mundo de los Bloques

Acciones:	Precond.	Añade-Efectos	Quita-Efectos
clear(X)	(on Y X) (cleartop Y)	(cleartop X)	(on Y X)
puton(X Y)	(cleartop Y) (cleartop X)	(on Y X)	(cleartop X)

Algunos Conceptos

- Una meta que se cumple se llama *protegida* (*protected*)
- Si para resolver una nueva meta, se requiere deshacer una meta *protegida*, se dice que se tiene una *violación de protección*
- En algunos tipos de violaciones de protección, un planeador lineal todavia puede solucionar el problema, ya que se deja al sistema en un estado donde las metas se pueden lograr
- El problema (sobretodo para los planeadores lineales) es que las metas son no-serializables (intercaladas)

Tipos de metas e interacciones

1. Destrucción temporal de submetas: “destrucción creativa”

(on A B)	(on A B)
(cleartop A)	(on B C)
(cleartop C)	

2. Interacción positiva: “usar objetos existentes”

(on C A)	(on A B)
(on D B)	(on B C)
(cleartop C)	(on C D)
(cleartop D)	

3. Intercambiar bloques: “doble cruce”

(on C A)	(above D A)
(on D B)	(above C B)
(cleartop C)	
(cleartop D)	

Interacciones

- Mucha investigación se enfocó a encontrar mecanismos para resolver estas interacciones
- Debido a los problemas de interacciones, los cuales complican demasiado el análisis, se hicieron varias simplificaciones:
  1. Estado estático del mundo
  2. Un solo agente
  3. Acciones discretas
  4. No translape (sólo una acción a la vez)
  5. Determinístico (efectos predicibles)

Consideraciones

Los planeadores más modernos han progresivamente considerado factores como:

- Acciones contínuas
- No se suponen que sean instantaneas
- Eventos no programados
- Acciones de agentes restringidas por intervalos de tiempo
- Agentes múltiples
- Incertidumbre de los efectos de las acciones
- ...

Planificadores

- *Planeador lineal*: Los pasos de planeación deben de realizarse en una secuencia ordenada
- *Planeadores no-lineales*: Los pasos están sólo parcialmente ordenados y su precedencia sólo se establece al final (*least commitment*)
- La idea es tratar de evitar violaciones a protecciones
- Remedios: poner la meta antes/después del intervalo, y buscar en el espacio de planes posibles

Planificadores

- En algunos tipos de interacción (e.g., doble cruce) ninguna opción resulta
- Otras complicaciones: *Truth dependency maintenance* (mantenimiento de verdad/consistencia)
- Los efectos de una tarea dependen en general del estado del mundo cuando se realiza la tarea, y este estado depende del orden de las tareas, el cual es conocido sólo parcialmente

Protección de suposición

1. La *protección de suposición* se refiere a que la suposición de verdad de q, que es prerequisito de una meta M, influye en los efectos de realizar M.
2. Si debido al reordenamiento de tareas, una meta P, que cambia la verdad de q, ocurre antes de M, los efectos de M pueden ser erroneos (i.e., las conclusiones de lo que va a pasar).
3. Los remedios son: poner la meta violada antes de la meta que determina la verdad de q, ponerla después de M, o volver a calcular los efectos de M.

Reducción de suposición

- La *reducción de suposición* es el supuesto que al escoger una reducción particular de una meta deseada en metas más primitivas permanece efectiva cuando el orden (tiempo) para realizar la tarea sea cambiado por un reordenamiento de tareas para resolver una protección de violación.
- Si la suposición de reducción es violada, la tarea debe de volverse a reducir.

Planeación

- No todas las interacciones son malas. Para aprovechar posibles interacciones positivas, hay que diferir lo más posible las instanciaciones de objetos
- E.g., si se quiere quitar bloque A de B para lograr (clear B), y una meta posterior requiere (on A C), es “posible” hacer los dos, si el objeto que especifica en donde colocar A se queda sin instanciar (como objetos anónimos) hasta que se vea que puede ponerse en C.

Componentes Escenciales

1. Mecanismo para representar y manejar la información del estado actual
2. Descripción de la meta o submetas a realizar
3. Conjunto de operadores primitivos
4. Conjunto de operadores con descripción de una secuencia de sub-metas (pueden especificar métodos para tratar fallas)
5. Mecanismo para aparear metas y operadores (asociación N a N)
6. Con lo anterior, el sistema tiene que planear en un modelo del mundo, y debe de poder obtener información de sus errores y exhibir aprendizaje

Preguntas a Resolver

1. ¿Cuál es una notación correcta para planes?
2. ¿Qué es un problema? ¿Qué significa que se pueda descomponer o reducir?
3. ¿Cómo se pueden anticipar y resolver violaciones de protección? ¿Cómo coordinar e introducir factores de uso/compartir recursos?
4. ¿Cómo mantener y producir un mapa del tiempo?
5. ¿Cuál es el espacio de búsqueda correcto: un espacio de posibles situaciones del mundo? ¿Un espacio de posibles planes? ¿Otra cosa? y (muy importante) ¿Cómo reducir este espacio y controlar la búsqueda? ¿Eficiente?
6. ¿Cómo y cuándo se traducen los planes en acciones en el mundo real? ¿Cómo mezclar planes/acciones?
7. ¿Cómo se manejan los errores al tiempo de planear/ejecutar? ¿Cómo se monitorea el progreso de un plan si las cosas salen mal?

Planeación

Meta principal: desarrollar un planificador independiente del dominio que maneje metas simultaneas
Componentes:

- Representación: De acciones, del mundo y objetos en él, y del tiempo
- Operadores de modificación de planes
- Estrategia global de búqueda

El planeador debe de facilitar la *reducción de búsqueda* y *control de interacciones*

Propiedades

- Un planeador es *correcto* si encuentra una solución al problema
- Un plan es *mínimo* si todas las acciones son necesarias
- Un plan en *óptimo* si no existe un plan más corto
- Un generador de planes es *completo* si eventualmente (rápido) genera un plan óptimo

Cronología

- General Problem Solver (GPS) - (57)
- STanford Research Institute Problem Solver (STRIPS) - (71)
- Otros: ABSTRIPS (Abstraction-Based STRIPS) - (73), HACKER (73), Warplan (74), Interplan (75)
- NOAH (Nets Of Action Hierarchies) - (75)
- Otros 2: NONLIN (77), NOAH distribuido, MOLGEN (81), SIPE (System for Interactive Planning and Execution monitoring) - (84), DEVISER - (83), ISI (84), O-Plan (85)
- GRAPHPLAN
- SATPLAN

GPS - Newell, Shaw and Simon (57)

- Primer programa en separar procedimientos generales de solución de conocimiento específico del dominio
- El espacio codificado en metas, objetos y operadores
- Uso de *Means-Ends analysis* (MEA): La idea es reducir la diferencia entre el estado actual y la meta
- El GPS usa las diferencias de dos formas:
  1. Sirve como una medida burda del progreso global hacia la meta (i.e., función de evaluación burda)
  2. Puede sugerir qué acción a realizar

GPS

Los operadores se describen como tripletas (generalmente por medio de una tabla):

1. Precondiciones
2. Función de transformación definiendo los efectos
3. Descripción de las diferencias reducidas por el operador

Algoritmo

Establece diferencia
Encuentra operador capaz de reducir la diferencia
IF operador puede aplicarse directamente, adelante
ELSE pon una submeta para lograr las condiciones necesarias para aplicar el operador
IF operador escogido falla, entonces *backtracking*

Usa *depth-first search*, pero ordena los operadores por reducción de diferencias

Suposiciones

1. El reducir la distancia implica que se va a poder llegar más fácil desde ahí (contra-ejemplo una colina intermedia)
2. El problema puede descomponerse en subproblemas más simples que pueden resolverse independientemente y luego combinarse

Ejemplo

Adaptado del libro de Winston: (Ex) Subc. Marcos invitado a Mexicali a dejar veladora

diferencias	procedimiento				
(km)	avion	tren	coche	taxi	pie
$D > 1000$	X				
$100 < D < 1000$		X	X		
$1 < D < 100$			X	X	
$D < 1$					X
	en avión	en tren	en coche	en taxi	–
	prerequisitos				

Deficiencia en manejo del modelo del mundo y cambios de estado (frame problem): construía un nuevo modelo del mundo para cada estado

STRIPS

STanford Research Institute Problem Solver (STRIPS) (71)
R. Fikes y N. Nilsson

- Usan *situational calculus*: Suponen que el agente es el único actor en un dominio estático
- En este esquema la generación de planes puede reducirse a un problema deductivo "sin tiempo"
- Dominio: navegación de un robot (*SHAKEY*) en un mundo con cuartos conectados con un conjunto de objetos (cajas) los cuales se les podía pedir que moviera
- Operadores aplicables para cada situación del modelo del mundo transformandolo en otro estado
- Meta: encontrar una secuencia de operadores que transforme el estado inicial a uno final

STRIPS

- Representación: El modelo del mundo dado como proposiciones lógicas
- Separación del probador de teoremas (para resolver preguntas dentro de un mundo dado - usan resolución) de la búsqueda (para ver qué operadores utilizar) basada en GPS-MEA
- Información inicial:
  - Modelo inicial del mundo (wff)
  - Operadores con precondiciones y efectos (wff schemata)
  - Meta (wff)

Algoritmo

G ← Primera meta en lista de metas
IF Satisface el modelo del mundo (M) a G
    IF G = última meta, Then éxito
    Else genera un nodo sucesivo (aplica op.)
Else encuentra diferencias
    Selecciona un nodo que tiene sucesores sin computar
    Selecciona un operador

Operadores

push(K,M,N)
- precondiciones: atr(M), at(K,M)
- delete list: atr(M), at(K,M).
- add list: atr(N), at(K,N)

pickup(X)
- precondiciones: emptyhand, clear(X), on(X,Y)
- delete list: emptyhand, clear(X), on(X,Y)
- add list: inhand(X)

putdown(Y)
- precondiciones: inhand(Y)
- delete list: inhand(Y)
- add list: emptyhand, clear(Y), on(Y,table)

unstack(U,Z)
- precondiciones: emptyhand, clear(U), on(U,Z)
- delete list: emptyhand, clear(U), on(U,Z)
- add list: inhand(U), clear(Z)

...

Ejemplo

el robot está en cuarto1
puerta12 conecta cuarto1 y cuarto2
cajaA está en cuarto2
puerta23 conecta cuarto2 y cuarto3
...

meta: llevar cajaA a cuarto3

Ejemplo

Strips busca la acción que queda más cerca de la meta:
e.g., empuja_por_puerta(cajaA,puerta23)
precondiciones:
a) el robot debe de estar en el mismo cuarto que la caja
b) el robot debe de estar junto a la caja
c) la puerta debe de estar abierta

Strips tendría que ver como satisfacer (a)

Strips aparte tenía sensores de visión y tacto para comparar el modelo del mundo con la realidad

Cálculo de Diferencias

- Si no se obtiene una contradicción (por el resolvedor de teoremas), se tiene una prueba incompleta (árbol con todos los decendientes hasta el momento)
- Entonces: fijarse en los operadores que tengan en sus *add-lists* las metas para continuar la prueba
- *Representación*: En lugar de tener una copia del mundo cada vez que se aplica un operador, se supone que el mundo no cambia al aplicar un operador (*suposición Strips*) e.g., cuartos, paredes, puertas y objetos permanecen...

Extensiones

- PLANEX = STRIPS + MACROPS o macro operadores (Fikes '72)
- Una vez encontrado un plan, lo generaliza lo más posible y lo guarda para su uso futuro
- Estas generalizaciones, llamadas MACROPs (macro-operadores) se pueden ver como acciones de alto nivel
- Cada sub-plan puede convertirse a su vez en un MACROP
- Se pueden resolver problemas similares a los vistos
- Los planes generalizados se guardan en tablas triangulares y se pueden usar fragmentos

Resumen

- STRIPS: General, no-jerárquico, lineal, restringido, un solo agente, suposición STRIPS (obviamente), *depth-first search*, ignora interacciones
- Búsqueda en espacio de situaciones (no de planes)
- Limitaciones para modificar un plan y atacar interacciones
- El tiempo y otros recursos no intervienen
- Ideas de Strips usadas en muchos de los planeadores siguientes

ABSTRIPS

- ABSTRIPS (Abstraction-Based STRIPS), E. Sacerdoti '73
- Idea: Planeación en una jerarquía de abstracciones en donde niveles sucesivos de detalle son introducidos
- Idea: Encontrar una solución en un espacio de abstracción y luego considerar los detalles del siguiente nivel
- Los espacios de abstracción difieren sólo en el nivel de detalle (i.e., el modelo no cambia) usado para especificar las precondiciones de los operadores (i.e., más precondiciones en niveles inferiores)

WARPLAN '74 Warren

- Primer planeador en Prolog
- Primero en no seguir una suposición lineal estricta
- Si un plan no podía encontrarse haciendo una concatenación de subplanes, se trataba de ordenar las acciones primitivas para no destruir secuencias de planes previamente construídas (*goal regression*)

Algoritmo

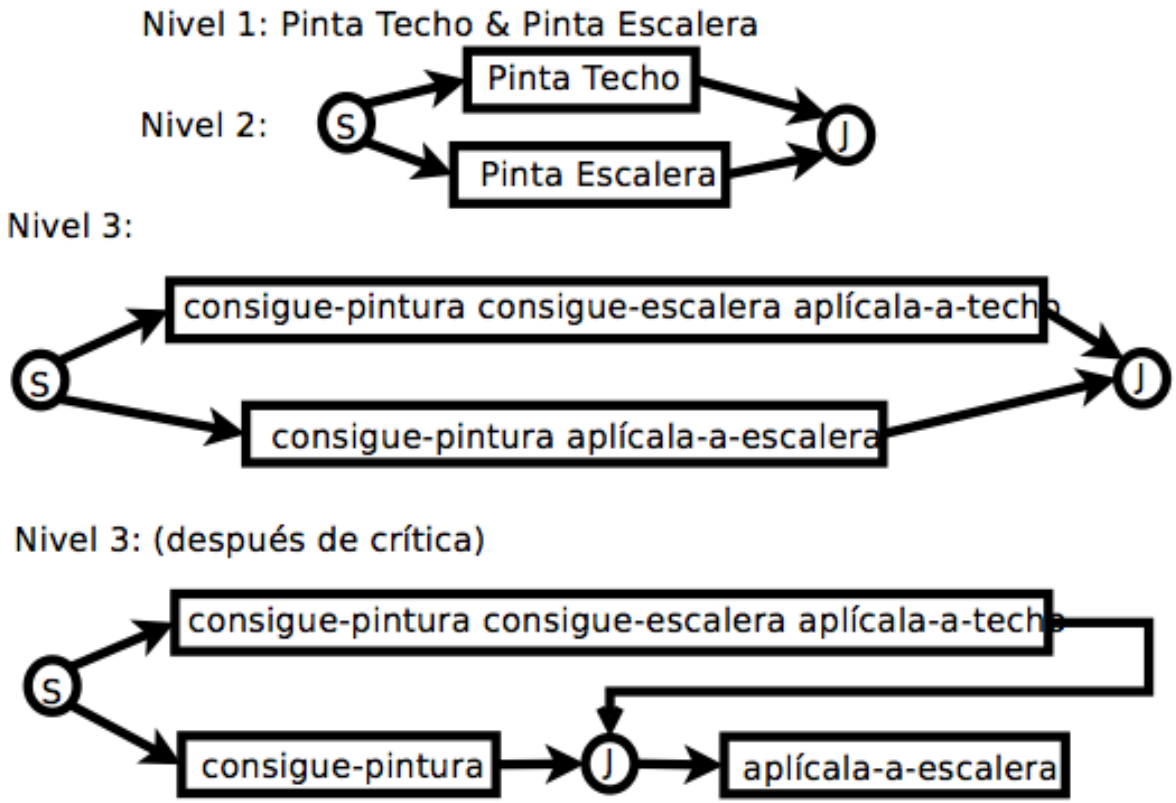
Intenta las metas de izquierda a derecha
IF se resuelve una meta, continua con la siguiente
ELSE busca un operador para resolverla
    IF el operador tiene conflicto con una parte anterior del plan
    THEN busca a la izquierda un lugar para ponerlo antes del punto del conflicto, cuidando que los pasos posteriores no borren los efectos del operador
    ELSE añade el operador al final
Una vez encontrado un lugar para el operador si sus precondiciones no se cumplen, ponlas como submetas

NOAH (Nets Of Action Hierarchies) '75 Sacerdoti

- Un plan se puede pensar como un orden parcial de acciones
- El hacer *backtracking* generalmente es porque se supuso un cierto orden
- La anomalía de Sussman puede resolverse fácilmente si los planes se representan como parcialmente ordenados
- Idea: Expandir diferentes partes del plan en paralelo lo más posible, y sólo imponer un orden cuando surge algún conflicto o cuando se tiene el plan completo (red)
- NOAH guarda el detalle del plan en todos los niveles junto con información de cómo cada nuevo nivel está relacionado con el anterior

NOAH

e.g., pintar el techo y pintar la escalera



NOAH

Mucho del trabajo se hace por medio de 5 críticas:

1. *Resuelve conflictos*: Busca en los *add/delete lists* para ver si existe algún conflicto (si una acción borra una precondición para otra meta). E.g. aplica pintura a escalera, tiene en su *delete-list* "tiene-escalera" que está en el *add-list* de "obten escalera"
2. *Usa objetos existentes*: Trata de evitar el introducir objetos extra innecesarios (hace sus instanciaciones los más tarde posible)
3. *Resuelve doble cruce*: Trata de indentificar el problema (restringido)
4. *Optimiza disjunciones*: Evalua alternativas para ver si alguna es mejor que otra
5. *Elimina precondiciones redundantes*: E.g., obten la pintura

NOAH

- No hace *backtracking*
- Supone que sólo existe una forma de expader una meta
- Cuando necesita resolver un conflicto, mueve la meta problemática al primer punto posterior (nunca a uno anterior) y selecciona el primero que encuentra (aunque puede ser desastroso)
- No preserva información de recursos al reordenar metas

Otros

- NONLIN (Tate '77): (i) guarda puntos de *backtracking*, corrige las interacciones, distingue efectos importantes
- NOAH distribuído (Corkill)
- SIPE (Wilkins '84): (i) permite inteacciones del usuario, (ii) razona con recursos
- DEVISER (Vere '83): Incluye tiempo (actividades asociadas con duración y tiempo inicial)
- ISIS (Fox y Smith '84): (i) Uso de recursos, (ii) restricciones de metas, físicas, causales y recursos
- O-Plan (Tate '85): Restricciones y manejo de tiempo
- MOLGEN (Stefik '81): Jerárquico + restricciones

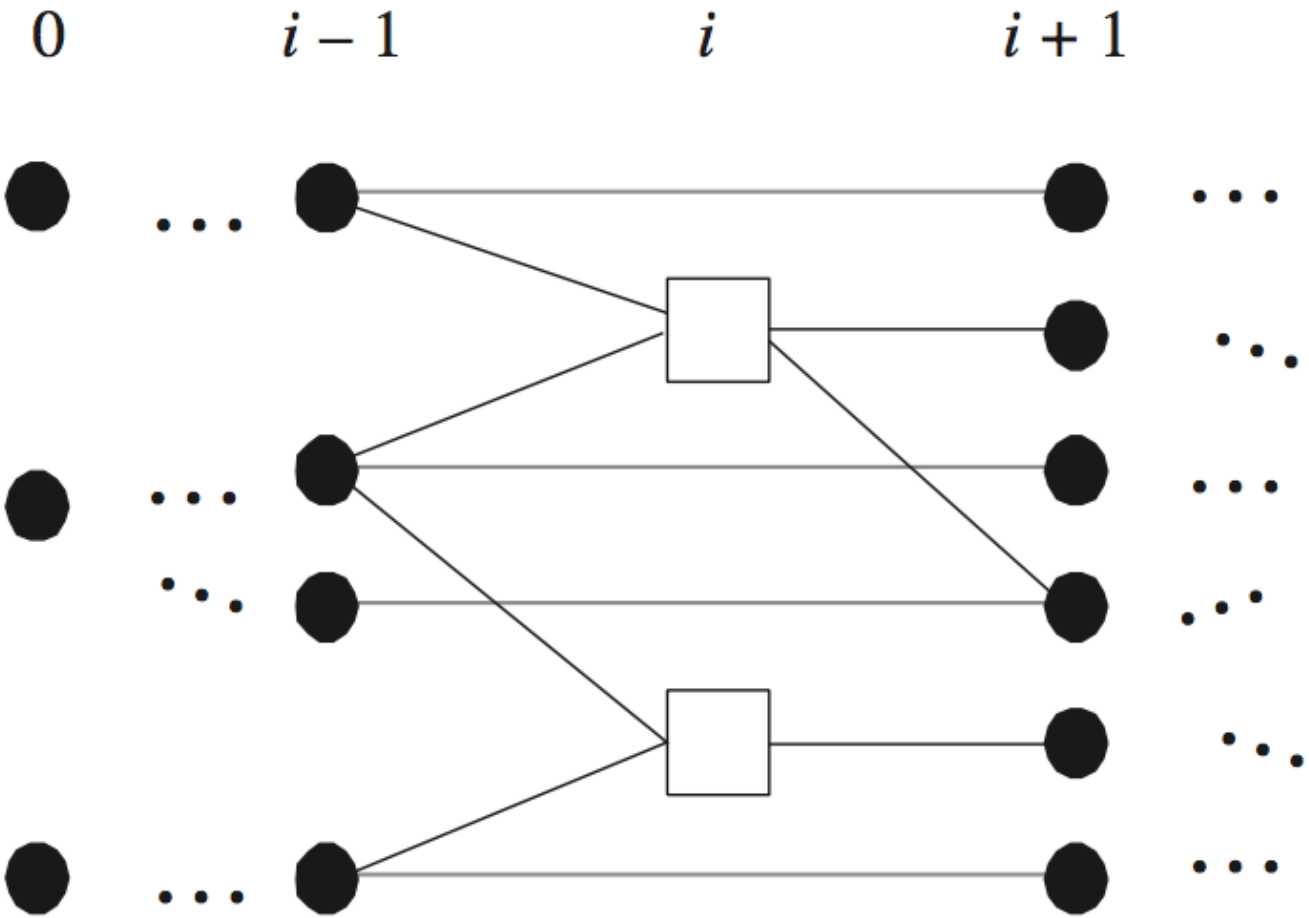
GRAPHPLAN (A. Blum y M. Furst - 95, 97)

- Es rápido y simple
- Su representación forma la base de muchos de los planificadores modernos
- Sigue dos pasos que se van alternando:
  1. Expansión del grafo
  2. Extracción de la solución

GRAPHPLAN - Expansión

- Se tienen dos tipos de nodos en niveles:
  1. Proposición (niveles par): A nivel cero están las condiciones iniciales.
  2. Acción (niveles impar): Existe un nodo por cada acción cuyas precondiciones estén en el nivel anterior y que sean consistentes
- Los arcos conectan nodos de proposiciones con acciones que los tengan como pre-condiciones y conectan nodos de acciones a proposiciones en el siguiente nivel que sean consecuencias de la acción

GRAPHPLAN

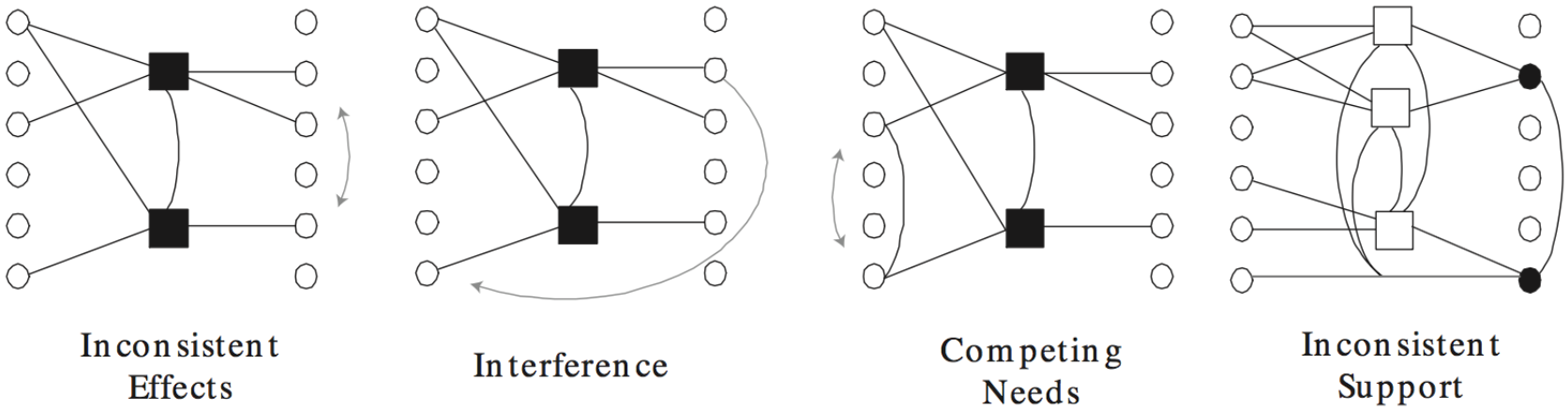


Mutex

Una parte central de GRAPHPLAN son las relaciones binarias mutuas de exclusión (*mutex*)

- Dos acciones en nivel *i* son *mutex* si:
  - El efecto de una acción es la negación del efecto de la otra acción (*inconsistent effects*)
  - Una acción elimina una pre-condición de la otra (*interference*)
  - Las acciones tienen precondiciones que son mutuamente exclusivas en el nivel anterior (*competing needs*)
- Dos proposiciones son *mutex* si (i) una es la negación de la otra o (ii) todas las formas de lograr la proposición (con acciones al nivel anterior) son *mutex* (*inconsistent support*).

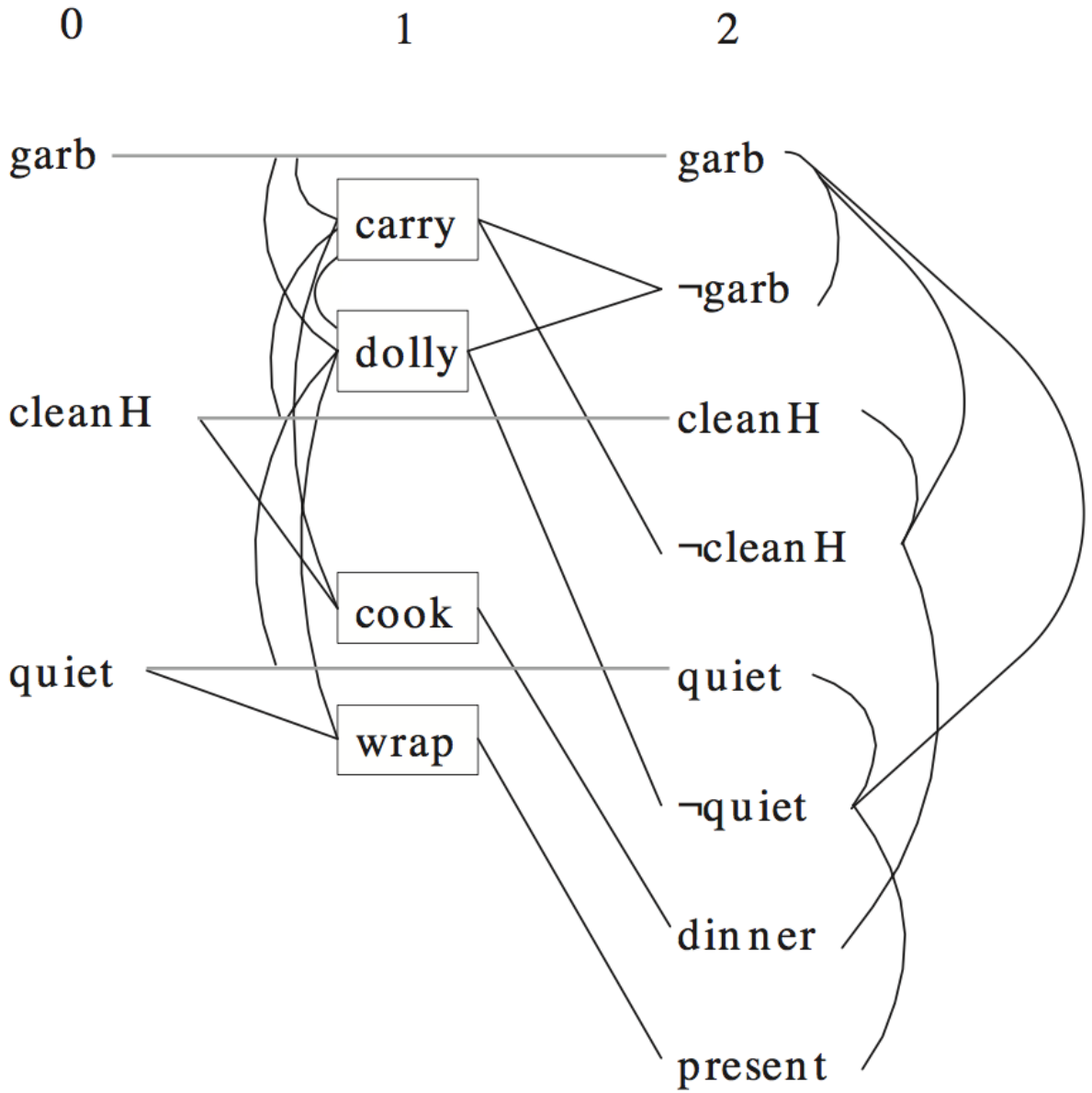
Descripción Gráfica de *Mutex*



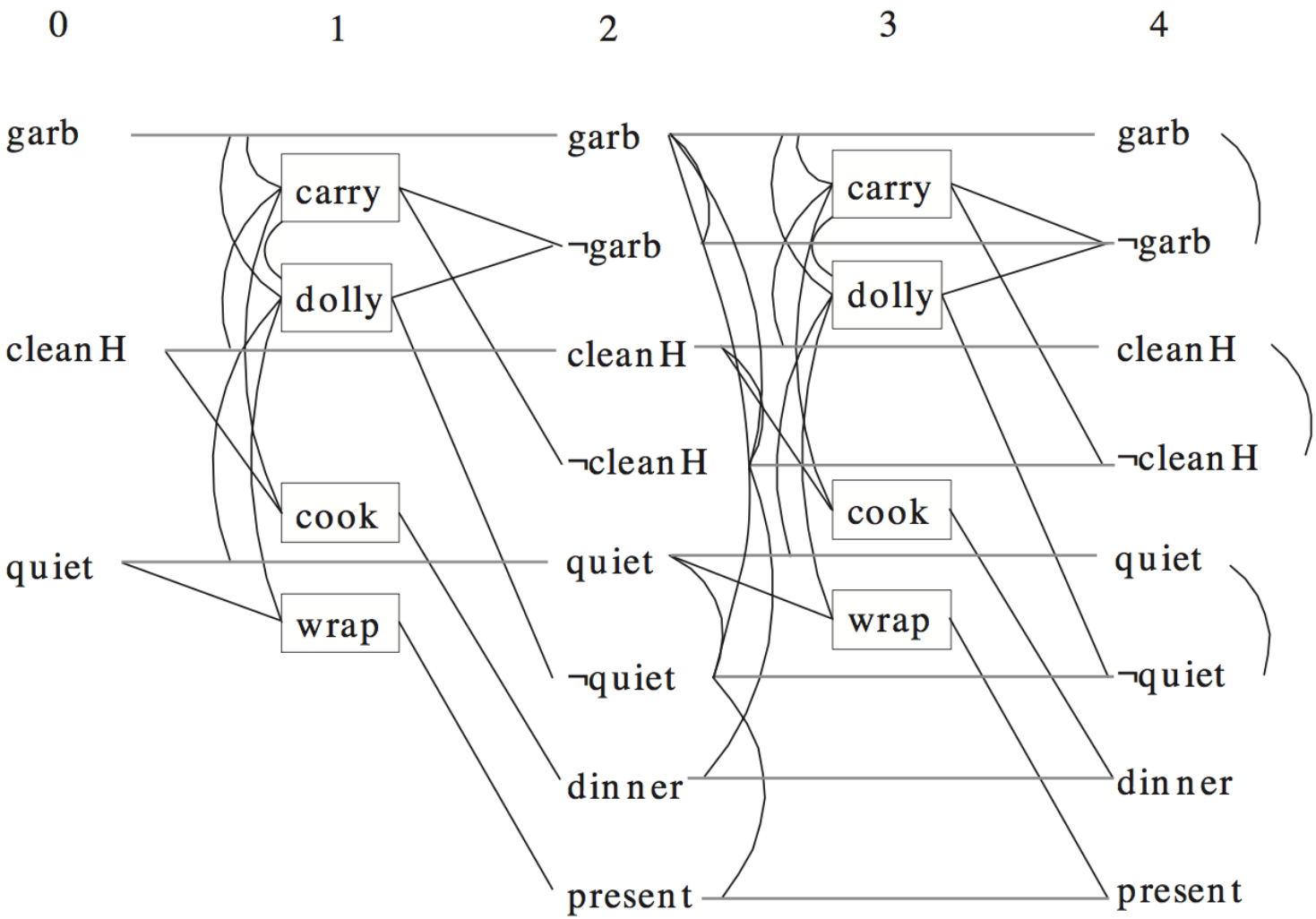
Ejemplo

```
Initial Conditions: (and (garbage) (cleanHands) (quiet))
Goal: (and (dinner) (present) (not (garbage)))
Actions:
      cook  :precondition (cleanHands)
            :effect (dinner)
      wrap  :precondition (quiet)
            :effect (present))
      carry :precondition
            :effect (and (not (garbage)) (not (cleanHands)))
      dolly :precondition
            :effect (and (not (garbage)) (not (quiet)))
```

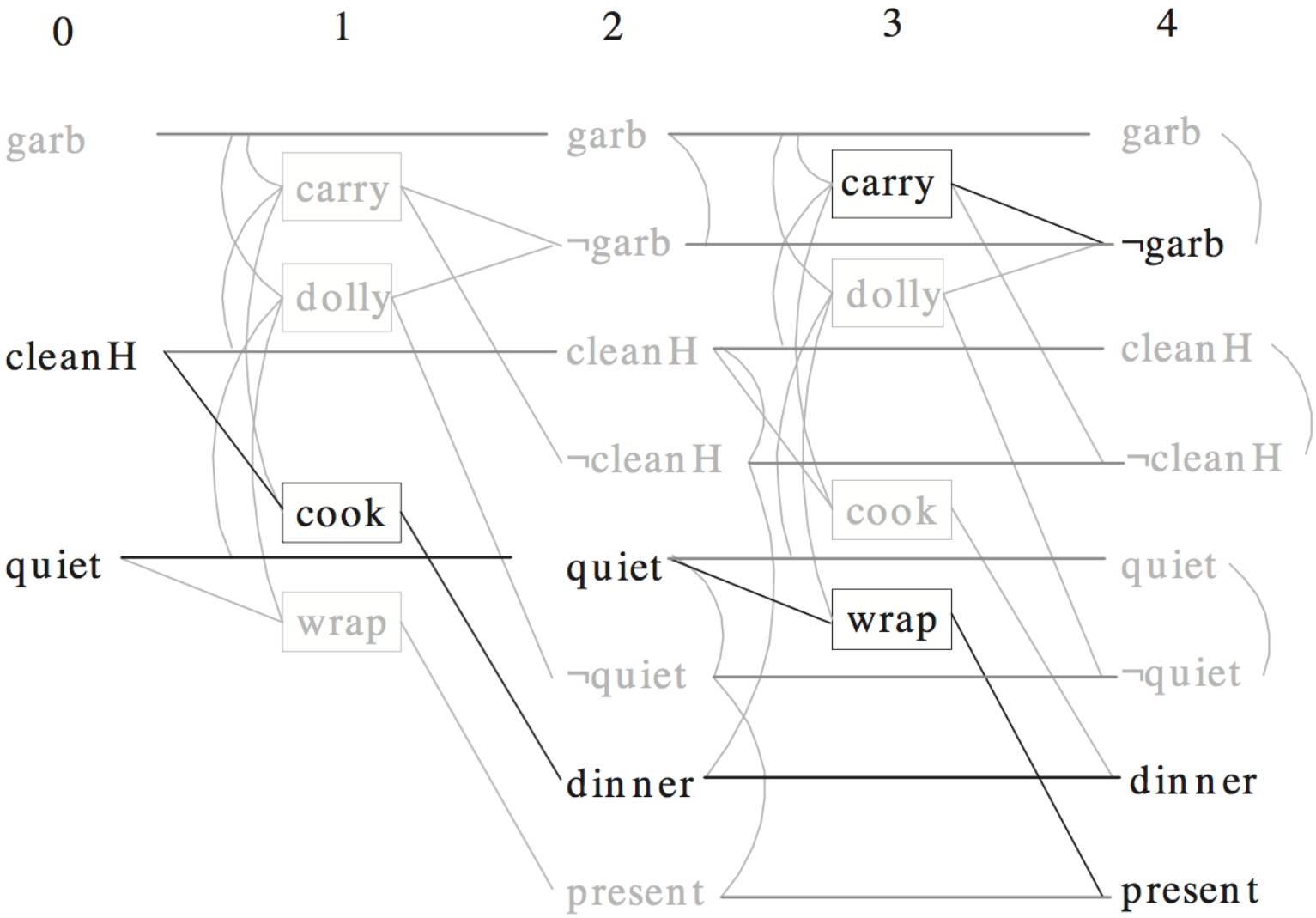
Ejemplo


0
1
2
garb
cleanH
quiet
carry
dolly
cook
wrap
garb
¬garb
cleanH
¬cleanH
quiet
¬quiet
dinner
present

Ejemplo


0
1
2
3
4
garb
cleanH
quiet
carry
dolly
cook
wrap
garb
¬garb
cleanH
¬cleanH
quiet
¬quiet
dinner
present
carry
dolly
cook
wrap
garb
¬garb
cleanH
¬cleanH
quiet
¬quiet
dinner
present

Ejemplo


0
1
2
3
4
garb
cleanH
quiet
carry
dolly
cook
wrap
garb
¬garb
cleanH
¬cleanH
quiet
¬quiet
dinner
present
carry
dolly
cook
wrap
garb
¬garb
cleanH
¬cleanH
quiet
¬quiet
dinner
present

GRAPHPLAN

- En el peor de los casos la expansión del grafo es polinomial pero la extracción de la solución es exponencial (aunque muchas veces domina la expansión)
- Existen ideas que se pueden aplicar de problemas de satisfacción de restricciones (CSP)
- Algunos ejemplos pueden ser el re-ordenamiento dinámico de variables (e.g., selecciona la variable con menos valores posibles)
- El algoritmo usaba *memoization* que significa guardar resultados parciales (re-utilizables)

GRAPHPLAN

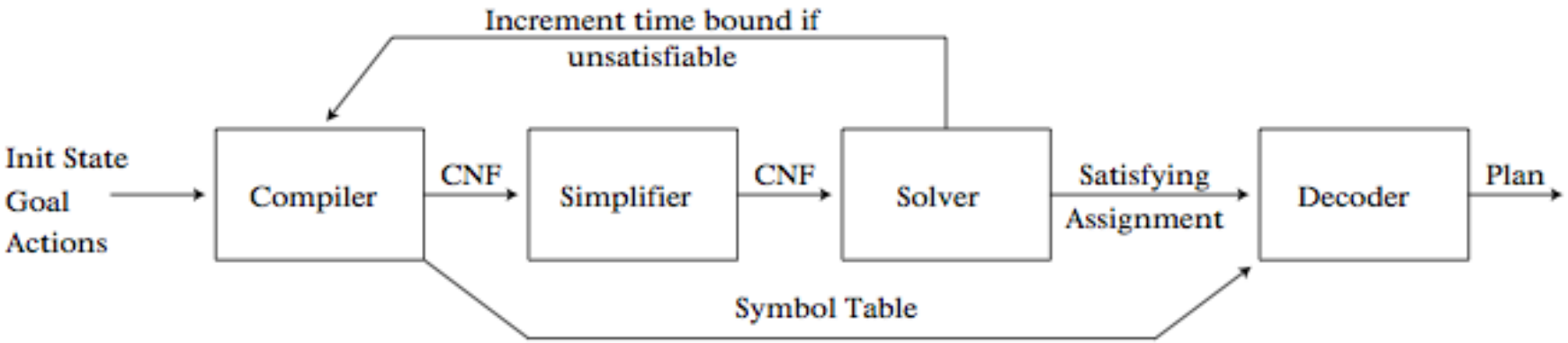
Se puede evitar trabajo adicional si se toma en cuenta que:

- Las proposiciones son monotónicamente crecientes (si aparecen en nivel i, también aparecen en nivel $i + 2$, etc.)
- Las acciones son monotónicamente crecientes
- Los *mutex* son monotónicamente decrecientes (un *mutex* entre acciones A y B en nivel i aparece en los niveles anteriores en donde estén A y B)
- Los *nogoods* (submetas que no se pueden cumplir) son monotónicamente decrecientes

SATPLAN

- Aunque en los inicios de la planeación se usaron probadores de teoremas generales, normalmente se hizo la suposición de que se necesitaban algoritmos de propósito específico
- Sin embargo, se empezaron a realizar avances muy importantes en la solución del problema de satisfación proposicional (*Propositional Sastisfiability Problem*) o SAT
- Algoritmo genérico: (i) Tomar un problema de planificación, (ii) transformarlo a un problema SAT, (iii) encontrar una solución y (iv) traducirlo de nuevo como solución al plan

SATPLAN


Increment time bound if unsatisfiable
Init State
Goal
Actions
Compiler
CNF
Simplifier
CNF
Solver
Satisfying Assignment
Decoder
Plan
Symbol Table

Algoritmo

Dado un problema de planeación:

- Adivina la longitud del plan
- Genera una fórmula lógica proposicional que al satisfacerse nos de la solución del plan
- Crea una tabla de símbolos (*symbol table*) que guarda las correspondencias entre variables proposicionales y la instancia de planeación
- Usa un simplificador para reducir la fórmula CNF (*Conjunctive Normal Form*)
- Busca una solución y tradúcela
- Si no se encuentra, crea una codificación de un plan más largo

¿Cómo Codificar?

Uno de los aspectos más importantes es cómo codificar el problema en una fórmula CNF

- Por convención, se usan sub-índices para denotar tiempo
- El estado inicial contiene todas las condiciones conocidas y las desconocidas se suponen falsas (*Closed Wold Assumotion*)
  E.g., $\text{garb}_0 \wedge \text{cleanH}_0 \wedge \text{quiet}_0 \wedge \neg\text{dinner}_0 \wedge \neg\text{present}_0$
- En la meta se pone lo se quiere cumplir en un tiempo *2n*
  E.g., $\neg\text{garb}_2 \wedge \text{dinner}_2 \wedge \text{present}_2$

¿Cómo Codificar?

- Acciones en los tiempos impares. Se supone que la acción en el tiempo t hace que se cumplan sus efectos en el tiempo $t+1$ dado que sus pre-condiciones se satisfacen en el tiempo $t-1$
  E.g., para la acción *cook* se tendría:
  $(\neg \text{cook}_1 \vee \text{dinner}_2) \wedge (\neg \text{cook}_1 \wedge \text{cleanH}_0)$
- Se tiene que definir cómo representar acciones cuando tienen variables
- Normalmente se añade información adicional
- Dado un grafo de planeación (construido usando GRAPHPLAN) se puede convertir en CNF.
  E.g., $\neg \text{garb}_4 \Rightarrow (\text{dolly}_3 \wedge \text{carry}_3 \wedge \text{maintain} - \text{no} - \text{garb}_3)$

Resolvedores SAT

- Sin un resolvedor SAT eficiente la compilación no sirve de nada
- En general existen dos tipos:
  1. Sistemáticos
  2. Estocásticos

Resolvedor SAT sistemático

- Lo más usado son variantes de DPLL (Davies, Logemann, Loveland; 1962)
- Ideas: Dada una CNF ϕ:
  1. Si una cláusula es una sola literal P (cláusula unitaria), ésta debe de ser verdadera
  2. Si cada vez que aparece una literal Q en ϕ tiene la misma polaridad (literal pura), se puede asignar su valor verdadero

Ejemplo

- $\phi = (A \vee B \vee \neg E) \wedge (B \vee \neg C \vee D) \wedge (\neg A) \wedge (B \vee C \vee E) \wedge (\neg D \vee \neg E)$
- $\neg A$ es una cláusula unitaria y B es una literal pura
- $\phi(\neg A) = (B \vee \neg E) \wedge (B \vee \neg C \vee D) \wedge (B \vee C \vee E) \wedge (\neg D \vee \neg E)$
- $\phi(B) = (\neg A) \wedge (\neg D \vee \neg E)$

DPLL

DPLL(ϕ)

If $\phi = \emptyset$ regresa SI

Else if hay una cláusula vacía en ϕ, regresa NO

Else if hay una literal pura $u \in \phi$,

regresa DPLL($\phi(u)$)

Else if si hay una cláusula unitaria $\{u\} \in \phi$,

regresa DPLL($\phi(u)$)

Else selecciona una variable $v \in \phi$

If DPLL($\phi(v)$) = SI, regresa SI

Else regresa DPLL($\phi(\neg v)$)

Resolvedor SAT Estocástico

- Son incompletos (no se sabe si fallan porque la fórmula es insatisfascible o porque no les alcanzó el tiempo)
- Pero son mucho más rápidos
- GSAT: *random-restart hill-climbing*. Selecciona una variable con base en el número de cláusulas satisfechas
- WALKSAT: Con probabilidad p selecciona la variable de GSAT, si no, selecciona otra aleatoriamente

Procesos de Decisión de Markov

- Existen problemas en que se tiene que decidir en cada estado la acción a realizar
- Este proceso de decisión secuenial se puede caracterizar como un proceso de decisión de Markov o MDP
- Un MDP modela un problema de decisión secuencial en donde el sistema evoluciona en el tiempo y es controlado por un agente
- La dinámica del sistema está determinada por una función de transición de probabilidad que mapea estados y acciones a otros estados

MDP

Formalmente, un MDP es una tupla $M =< S, A, \Phi, R >$ formada por:

- Un conjunto finito de estados S, $(s_i \in S, i = \{1, \ldots, n\})$
- Un conjunto finito de acciones A, que pueden depender de cada estado $(a_j(s_i), j = \{1, \ldots, m\})$
- Una función de recompensa (R), que define la meta y mapea cada estado–acción a un número (recompensa), indicando lo deseable del estado $(f(s, a) \Rightarrow \mathcal{R})$
- Un modelo del ambiente o función de transición de estados $\Phi(s'|s, a)$ $(\Phi : A \times S \rightarrow S)$ que nos dice la probabilidad de alcanzar el estado $s' \in S$ al realizar la acción $a \in A$ en el estado $s \in S$

Elementos Adicionales

- Política (π): define cómo se comporta el sistema en cierto tiempo. Es un mapeo (a veces estocástico) de los estados a las acciones ($\pi(S) \rightarrow A$)
- Función de valor (V): indica lo que es bueno a largo plazo. Es la recompensa total que un agente puede esperar acumular empezando en un estado s ($V(s)$) o en un estado haciendo una acción a ($Q(s, a)$)
- Las recompensas están dadas por el ambiente, pero los valores se deben de estimar (aprender) con base en las observaciones

Modelos de Recompensas

- Dado un estado $s_t \in S$ y una acción $a_t \in \mathcal{A}(s_t)$, el agente recibe una recompensa r_{t+1} y se mueve a un nuevo estado s_{t+1}
- Si las recompensas recibidas después de un tiempo t se denotan como: r_{t+1}, r_{t+2}, r_{t+3}, ..., lo que queremos es maximizar lo que esperamos recibir de recompensa total acumulada (R_t)
- Si se tiene un punto terminal se llaman tareas *episódicas*, si no se tiene se llaman tareas *contínuas*.

Modelos de Recompensas

- **Horizonte finito**: El agente trata de optimizar su recompensa esperada en los siguientes h pasos, sin preocuparse de lo que ocurra despues:

$$E(\sum_{t=0}^{h} r_t)$$

- Se puede usar como:
  - *Política no estacionaria*: En el primer paso se toman los h siguientes pasos, en el siguiente los $h-1$, etc., hasta terminar. El problema principal es que no siempre se conoce cuántos pasos considerar
  - *Receeding-horizon control*: Siempre se toman los siguientes h pasos

Modelos de Recompensas

- **Horizonte infinito** (la más utilizada): Las recompensas que recibe un agente son reducidas geométricamente de acuerdo a un factor de descuento γ ($0 \leq \gamma \leq 1$):

$$R = r_0 + \gamma r_1 + \gamma^2 r_2 + \ldots = \sum_{t=0}^{\infty} \gamma^k r_t$$

donde γ se conoce como la *razón de descuento* y lo que queremos maximizar es la recompensa total esperada:

$$E(\sum_{t=0}^{\infty} \gamma^t r_t)$$

Modelos de Recompensas

- **Recompensa promedio**: Optimizar a largo plazo la recompensa promedio:

$$lim_{h\to\infty} E(\frac{1}{h}\sum_{t=0}^{h} r_t)$$

Problema: No distingue políticas que reciban grandes recompensas al principio de las que no.

Modelo Markoviano

- Se supone que se cumple con la propiedad Markoviana (las transiciones de estado sólo dependen del estado actual) y las probabilidades de transición están dadas por:

$$\mathcal{P}^a_{ss'} = Pr\{s_{t+1} = s' \mid s_t = s, a_t = a\}$$

  El valor de recompensa esperado es:

$$\mathcal{R}^a_{ss'} = E\{r_{t+1} \mid s_t = s, a_t = a, s_{t+1} = s'\}$$

- Lo que se busca es estimar las funciones de valor. Esto es, qué tan bueno es estar en un estado (o realizar una acción)
- La noción de "qué tan bueno" se define en términos de recompensas futuras o recompensas esperadas

Funciones de Valor

- La política π es un mapeo de cada estado $s \in S$ y acción $a \in \mathcal{A}(s)$ a la probabilidad $\pi(s, a)$ de tomar la acción a estando en el estado s
- El valor de un estado s bajo la política π, denotado como $V^\pi(s)$, es la recompensa total esperada estando en el estado s y siguiendo la política π:

$$V^\pi(s) = E_\pi\{R_t \mid s_t = s\} = E_\pi\left\{\sum_{k=o}^{\infty} \gamma^k r_{t+k+1} \mid s_t = s\right\}$$

- El valor esperado tomando una acción a en estado s bajo la política π ($Q^\pi(s, a)$):

$$\begin{aligned} Q^\pi(s, a) &= E_\pi\{R_t \mid s_t = s, a_t = a\} \\ &= E_\pi\left\{\sum_{k=o}^{\infty} \gamma^k r_{t+k+1} \mid s_t = s, a_t = a\right\} \end{aligned}$$

Funciones de Valor Óptimas

- Las funciones de valor óptimas se definen como:

$$V^*(s) = max_\pi V^\pi(s) \text{ y } Q^*(s,a) = max_\pi Q^\pi(s,a)$$

- Las cuales se pueden expresar como las ecuaciones de optimalidad de Bellman:

$$V^*(s) = max_a \sum_{s'} \mathcal{P}^a_{ss'}[\mathcal{R}^a_{ss'} + \gamma V^*(s')]$$

$$Q^*(s,a) = \sum_{s'} \mathcal{P}^a_{ss'}[\mathcal{R}^a_{ss'} + \gamma V^*(s')]$$

$$Q^*(s,a) = \sum_{s'} \mathcal{P}^a_{ss'}[\mathcal{R}^a_{ss'} + \gamma max_{a'} Q^*(s',a')]$$

Métodos de Solución

Existen tres métodos principales de resolver MDPs:

1. Programación Dinámica
2. Monte Carlo, y
3. Diferencias Temporales o de Aprendizaje por Refuerzo

Programación Dinámica

- Si se conoce el modelo del ambiente, o sea, las transiciones de probabilidad ($\mathcal{P}^a_{ss'}$) y los valores esperados de recompensas ($\mathcal{R}^a_{ss'}$), las ecuaciones de optimalidad de Bellman nos representan un sistema de $|S|$ ecuaciones y $|S|$ incognitas
- Consideremos primero como calcular la función de valor V^π dada una política arbitraria π.

Funciones de Valor para una Política

$$
\begin{aligned}
V^{\pi}(s) &= E_{\pi}\{R_t \mid s_t = s\} \\
&= E_{\pi}\left\{r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \ldots \mid s_t = s\right\} \\
&= E_{\pi}\left\{r_{t+1} + \gamma V^{\pi}(s_{t+1}) \mid s_t = s\right\} \\
&= \sum_a \pi(s,a) \sum_{s'} \mathcal{P}^a_{ss'}[\mathcal{R}^a_{ss'} + \gamma V^{\pi}(s')]
\end{aligned}
$$

donde $\pi(s,a)$ es la probabilidad de tomar la acción a en estado s bajo la política π.

Funciones de Valor para una Política

- Podemos hacer aproximaciones sucesivas, evaluando $V_{k+1}(s)$ en términos de $V_k(s)$.

$$V_{k+1}(s) = \sum_a \pi(s,a) \sum_{s'} \mathcal{P}^a_{ss'} [\mathcal{R}^a_{ss'} + \gamma V_k(s')]$$

- Podemos entonces definir un algoritmo de evaluación iterativa de políticas

Funciones de Valor para una Política

Inicializa $V(s) = 0$ para toda $s \in S$

Repite

$\Delta \leftarrow 0$

Para cada $s \in S$

$v \leftarrow V(s)$

$V(s) \leftarrow \sum_a \pi(s,a) \sum_{s'} \mathcal{P}^a_{ss'} [\mathcal{R}^a_{ss'} + \gamma V(s')]$

$\Delta \leftarrow max(\Delta, |v - V(s)|)$

Hasta que $\Delta < \theta$ (número positivo pequeño)

Regresa $V \approx V^\pi$

Iteración de Políticas

- Calculamos la función de valor de una política para tratar de encontrar mejores políticas
- Dada una función de valor, podemos probar una acción $a \neq \pi(s)$ y ver si su $V(s)$ es mejor o peor que el $V^{\pi}(s)$
- En lugar de hacer un cambio en un estado y ver el resultado, se pueden considerar cambios en todos los estados considerando todas las acciones de cada estado, seleccionando aquella que parezca mejor de acuerdo a una política *greedy*.
- Podemos entonces calcular una nueva política $\pi'(s) = \text{argmax}_a Q^{\pi}(s, a)$ y continuar hasta que no mejoremos.

Iteración de Políticas

- Esto sugiere, partir de una política (π_0) y calcular la función de valor (V^{π_0}), con la cual encontrar una mejor política (π_1) y así sucesivamente hasta converger a π^* y V^*.
- A este procedimiento se llama iteración de políticas

Iteración de Políticas

$V(s) \in \mathcal{R}$ y $\pi(s) \in \mathcal{A}(s)$ arbitrariamente $\forall s \in S$ (**Inicializa**)
Repite (**Evaluación de Política**)
    $\Delta \leftarrow 0$
    Para cada $s \in S$
        $v \leftarrow V(s)$
        $V(s) \leftarrow \sum_{s'} \mathcal{P}_{ss'}^{\pi(s)}[\mathcal{R}_{ss'}^{\pi(s)} + \gamma V(s')]$
        $\Delta \leftarrow max(\Delta, |v - V(s)|)$
Hasta que $\Delta < \theta$ (número positivo pequeño)
pol-estable $\leftarrow$ true (**Mejora de Política**)
Para cada $s \in S$:
    $b \leftarrow \pi(s)$
    $\pi(s) \leftarrow \text{argmax}_a \sum_{s'} \mathcal{P}_{ss'}^{a}[\mathcal{R}_{ss'}^{a} + \gamma V(s')]$
    if $b \neq \pi$, then *pol-estable* $\leftarrow$ false
If *pol-estable*, then stop, else evalúa nva. política

Iteración de Valor

- Iteración de políticas en cada iteración evalúa la política y requiere recorrer todos los estados varias veces
- El paso de evaluación de política lo podemos truncar sin perder la garantía de convergencia, después de recorrer una sola vez todos los estados
- A esta forma se le llama iteración de valor (*value iteration*) y se puede escribir combinando la mejora en la política y la evaluación de la política truncada como sigue:

$$V_{k+1}(s) = max_a \sum_{s'} \mathcal{P}^a_{ss'}[\mathcal{R}^a_{ss'} + \gamma V_k(s')]$$

- Se puede ver como expresar la ecuación de Bellman en una regla de actualización

Iteración de Valor

Inicializa $V(s) = 0$ para toda $s \in S$
Repite
$\Delta \leftarrow 0$
Para cada $s \in S$
$v \leftarrow V(s)$
$V(s) \leftarrow max_a \sum_{s'} \mathcal{P}^a_{ss'}[\mathcal{R}^a_{ss'} + \gamma V^*(s')]$
$\Delta \leftarrow max(\Delta, |v - V(s)|)$
Hasta que $\Delta < \theta$ (número positivo pequeño)
Regresa una política determinística tal que:
$\pi(s) = argmax_a \sum_{s'} \mathcal{P}^a_{ss'}[\mathcal{R}^a_{ss'} + \gamma V^*(s')]$

Monte Carlo

- Los métodos de Monte Carlo, sólo requieren de experiencia y la actualización se hace por episodio más en lugar de en cada paso
- El valor de un estado es la recompensa esperada que se puede obtener a partir de ese estado
- Para estimar V^{π} y Q^{π} podemos tomar estadísticas haciendo un promedio de las recompensas obtenidas

Monte Carlo para Estimar V^{π}

Repite
Genera un episodio usando π
Para cada estado s en ese episodio:
    $R \leftarrow$ recompensa después de la primera ocurrencia de s
    Añade R a $recomp(s)$
    $V(s) \leftarrow$ promedio($recomp(s)$)

Monte Carlo

- Para estimar pares estado-acción (Q^{π}) corremos el peligro de no ver todos los pares, por lo que se busca mantener la exploración.
- Lo que normalmente se hace es considerar sólo políticas estocásticas que tienen una probabilidad diferente de cero de seleccionar todas las acciones.

Monte Carlo para Mejorar Políticas

- Con Monte Carlo podemos alternar entre evaluación y mejoras con base en cada episodio
- La idea es que después de cada episodio las recompensas observadas se usan para evaluar la política y la política se mejora para todos los estados visitados en el episodio

Algoritmo de Monte Carlo para Mejorar Políticas

Repite
Genera un episodio usando π con exploración
Para cada par (s, a) en ese episodio:
    $R \leftarrow$ recompensa después de la primera ocurrencia de (s, a)
    Añade R a $recomp(s, a)$
    $Q(s, a) \leftarrow$ promedio$(recomp(s, a))$
Para cada s en el episodio:
    $\pi(s) \leftarrow \text{argmax}_a Q(s, a)$

Diferencias Temporales

- Los métodos de TD combinan las ventajas de los dos anteriores: permiten hacer *bootstrapping* - estimar valores con base en otras estimaciones - (como DP) y no requieren tener un modelo del ambiente (como MC).
- Métodos tipo TD sólo tienen que esperar el siguiente paso.
- TD usan el error o diferencia entre predicciones sucesivas (en lugar del error entre la predicción y la salida final) aprendiendo al existir cambios entre predicciones sucesivas.

Esquemas de Exploración

- $\epsilon-$*greedy*: La mayor parte del tiempo se selecciona la acción que da el mayor valor estimado, pero con probabilidad ϵ se selecciona una acción aleatoriamente.
- *softmax*: La probabilidad de selección de cada acción depende de su valor estimado. La más común sigue una distribución de Boltzmann o de Gibbs, y selecciona una acción con la siguiente probabilidad:

$$\frac{e^{Q_t(a)/\tau}}{\sum_{b=1}^{n} e^{Q_t(b)/\tau}}$$

donde τ es un parámetro positivo (temperatura).

Algoritmos "on" y "off-policy"

- Los algoritmos *on-policy*: Estiman el valor de la política mientras la usan para el control. Se trata de mejorar la política que se usa para tomar decisiones.
- Los algoritmos *off-policy*: Usan la política y el control en forma separada. La estimación de la política puede ser por ejemplo *greedy* y la política de comportamiento puede ser ϵ-*greedy*.
- Esto es, la política de comportamiento está separada de la política que se quiere mejorar (es lo que hace Q-learning)

Aprendizaje por Refuerzo

- Los algoritmos son incrementales y fácil de computar
- Actualizan las funciones de valor usando el error entre lo estimado y la suma de la recompensa inmediata y lo estimado del siguiente estado
- El más simple TD(0) es:

$$V(s_t) \leftarrow V(s_t) + \alpha \left[r_{t+1} + \gamma V(s_{t+1}) - V(s_t) \right]$$

Algoritmo TD(0)

Inicializa $V(s)$ arbitrariamente y π a la política a evaluar
Repite (para cada episodio):
    Inicializa s
    Repite (para cada paso del episodio):
        $a \leftarrow$ acción dada por π para s
        Realiza acción a; observa la recompensa, r,
            y el siguiente estado, s'
        $V(s) \leftarrow V(s) + \alpha\,[r + \gamma V(s') - V(s)]$
        $s \leftarrow s'$
    hasta que s sea terminal

SARSA

- La actualización de valores tomando en cuenta la acción sería:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)]$$

- y el algoritmo es prácticamente el mismo, sólo que se llama SARSA

Algoritmo SARSA

Inicializa $Q(s, a)$ arbitrariamente
Repite (para cada episodio):
    Inicializa s
    Selecciona una a a partir de s usando la política
        dada por Q (e.g., ϵ–greedy)
    Repite (para cada paso del episodio):
        Realiza acción a, observa r, s'
        Escoge a' de s' usando la política derivada de Q
        $Q(s, a) \leftarrow Q(s, a) + \alpha\,[r + \gamma Q(s', a') - Q(s, a)]$
        $s \leftarrow s'$; $a \leftarrow a'$;
    hasta que s sea terminal

Q-Learning

- Uno de los desarrollos más importantes en aprendizaje por refuerzo fué el desarrollo de un algoritmo "fuera-de-política" (*off-policy*) conocido como Q-learning.
- La idea principal es realizar la actualización de la siguiente forma (Watkins, 89):

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_{t+1} + \gamma max_a Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)]$$

Algoritmo Q-Learning

Inicializa $Q(s, a)$ arbitrariamente
Repite (para cada episodio):
    Inicializa s
    Repite (para cada paso del episodio):
        Selecciona una a de s usando la política dada por Q (e.g., ϵ–greedy)
        Realiza acción a, observa r, s'
        $Q(s, a) \leftarrow Q(s, a) + \alpha \left[r + \gamma max'_a Q(s', a') - Q(s, a) \right]$
        $s \leftarrow s'$;
    hasta que s sea terminal

Estrategias para Espacios Grandes

- Uno de los problemas principales de RL es su aplicación a espacios grandes (muchos estados y acciones). Aunque los algoritmos convergen en teoría, en la práctica pueden tomar un tiempo inaceptable.
- Se han propuesto diferentes estrategias para esto:
  1. Actualizar varias funciones de valor a la vez
  2. Usar aproximación de funciones
  3. Aprender un modelo y usarlo
  4. Utilizar abstracciones y jerarquías
  5. Incorporar ayuda adicional