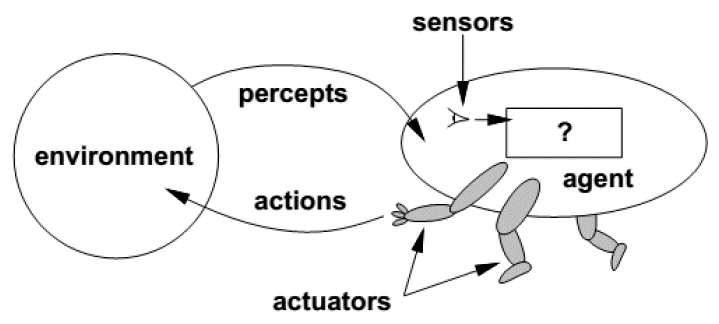
**Inteligencia artificial**

|  |  |
| --- | --- |
| **Sistemas que piensan como humanos** | **Sistemas que piensan racionalmente** |
| Se refiere a la automatización de actividades que vinculamos con procesos de pensamiento humano, actividades como la toma de decisiones, resolución de problemas, aprendizaje.  Se intenta logar que los computadores piensen. Máquinas con mentes, en el más amplio sentido literal. | Es el estudio de las facultades mentales mediante el uso de modelos computacionales.  El estudio de los cálculos que hacen posible percibir, razonar y actuar. |
| **Sistemas que actúan como humanos** | **Sistemas que actúan racionalmente** |
| Hace referencia a desarrollar máquinas con capacidad para realizar funciones que cuando son realizadas por personas requieren de inteligencia.  El estudio de cómo lograr que los computadores realicen tareas que, por el momento, los humanos hacen mejor. | Es el estudio del diseño de agentes inteligentes. Está relacionada con conductas inteligentes en artefactos. |

Se puede dividir el concepto de inteligencia artificial según 4 tipos de sistemas:

Las definiciones de la izquierda hacen hincapié en la forma de actuar de los humanos, mientras que las de la derecha toman como referencia un concepto ideal de inteligencia, llamado racionalidad. Un sistema es racional si hace ‘lo correcto’, en función de su conocimiento.

El enfoque centrado en el comportamiento humano debe ser una ciencia empírica, que incluya hipótesis y confirmaciones mediante experimentos. El enfoque racional implica una combinación de matemáticas e ingeniería. Cada grupo al mismo tiempo ha ignorado y ha ayudado al otro.

**Agentes inteligentes**

Un **agente** es cualquier cosa capaz de percibir su **ambiente** con la ayuda de **sensores** y actuar en ese medio utilizando **actuadores**.

**Percepciones**

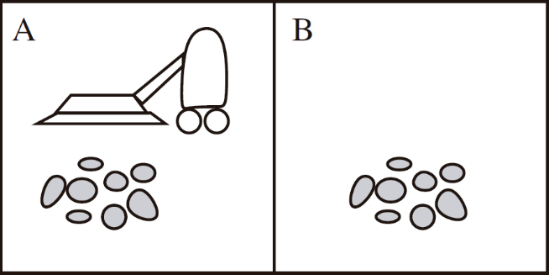
* Son entradas que recibe el agente en cualquier instante. Son imágenes de cómo está el ambiente en un determinado momento.
* La secuencia de percepciones refleja el historial completo de lo que el agente ha recibido. En general, un agente tomará una decisión en un momento dado dependiendo de la secuencia completa de percepciones hasta ese instante.

Función de Agente y Programa Agente

* En términos matemáticos se puede decir que el comportamiento del agente viene dado por la función del agente que mapea una secuencia de percepciones con una acción. La implementación de esta función de agente se da por medio del programa de agente que se ejecuta en una arquitectura dada.
* La función del agente es una descripción matemática abstracta; el programa del agente es una implementación completa que se ejecuta sobre la arquitectura del agente.

Ejemplo Aspiradora:

*Percepts*: lugar y contenido. Ej.: [A, Dirty]

*Actions*: Left, Right, Suck, NoOp

|  |  |
| --- | --- |
| **Percept sequence** | **Action** |
| [A, Clean] | Right |
| [A, Dirty] | Suck |
| [B, Clean] | Left |
| [B, Dirty] | Suck |
| [A, Clean], [A, Clean] | Right |
| [A, Clean], [A, Dirty] | Suck |
| … | … |

**Agente Racional**

Racionalidad

Es una medida de performance que indica el grado de éxito de un agente. Se mide en base a los estados del ambiente. Depende de:

* La medida de rendimiento que define el criterio de éxito (performance).
* El conocimiento previo del medio que habita el agente.
* Las acciones que el agente puede realizar.
* La secuencia de percepciones del agente hasta el momento.

Definición de agente racional

Un agente racional es aquel que hace lo correcto. Para cada posible secuencia de percepciones, un agente racional debe elegir una acción que se espera que maximice su medida de rendimiento, basándose en las evidencias provistas por la secuencia de percepciones y cualquier conocimiento previo que el agente posea.

Omnisciencia, aprendizaje y autonomía

* Omnisciencia: un agente omnisciente sabe cuál va a ser el resultado de sus acciones. Es distinto de Racionalidad. La omnisciencia, en realidad, no es posible.
* Recopilación de información: llevar a cabo acciones con la intención de modificar percepciones futuras.
* Aprendizaje: es la capacidad de generar nuevo conocimiento a partir de las percepciones. Se espera que un agente racional sea capaz de aprender.
* Autonomía: es la capacidad del agente de no depender solamente de conocimientos previos. El agente debe saber aprender a determinar cómo tiene que compensar el conocimiento incompleto inicial.

**Naturaleza de los ambientes**

Para diseñar un agente siempre se debe especificar su entorno de trabajo, denominado PEAS o REAS (Rendimiento, Entorno, Actuadores, Sensores).

Sirve para definir el ambiente donde el agente va situarse. En el diseño de un agente, el primer paso debe ser siempre especificar el entorno de trabajo de la forma más completa posible.

Ejemplo Taxista automático:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Agente** | **Medidas de Rendimiento** | **Entorno** | **Actuadores** | **Sensores** |
| Taxista | Seguro, rápido, legal, viaje confortable, maximización del beneficio. | Carreteras, otro tráfico, peatones, clientes. | Dirección, acelerador, freno, señal, bocina, visualizador. | Cámaras, sónar, velocímetro, GPS, tacómetro, visualizador de la aceleración, sensores del motor, teclado. |

**Clasificaciones de los entornos de trabajo**

* **Completamente observable vs parcialmente observable**

Completamente observable:

* Los sensores del agente le proporcionan acceso al estado completo del medio en cada momento.
* Los sensores detectan todos los aspectos que son relevantes en la toma de decisiones; la relevancia, en cada momento, depende de las medidas de rendimiento.
* Este tipo de entorno es conveniente ya que el agente no necesita mantener ningún estado interno para saber qué sucede en el mundo.

Parcialmente observable o no observable: Se da cuando los sensores no reciben información de parte del sistema, cuando son poco exactos o debido al ruido.

* **Un único agente vs multiagente**
* Ejemplo único agente: un agente resolviendo un crucigrama por sí mismo. Puede haber más agentes, pero no influyen.
* Ejemplo multiagente: un agente que juega al ajedrez.
* No siempre es fácil distinguir entre un entorno multiagente o con un único agente. Hay que saber distinguir qué entidades deben considerarse agentes.
* **Determinístico vs estocástico**

Entorno determinista

* Se da cuando el siguiente estado del medio está totalmente determinado por el estado actual y la acción ejecutada por el agente (ej.: aspiradora). El agente asegura cuál es el resultado de la acción. De lo contrario, es estocástico.
* Si el medio es totalmente observable y determinista, un agente no se tiene que preocupar de la incertidumbre. Si el medio es parcialmente observable, puede parecer estocástico.
* Si el medio es determinista, excepto para las acciones de otros agentes, decimos que el medio es estratégico.
* **Episódico vs secuencial**

Entorno episódico

* La experiencia del agente se divide en episodios atómicos. Cada episodio consiste en la percepción del agente y la realización de una única acción posterior.
* El siguiente episodio no depende de las acciones que se realizaron en episodios previos. La elección de la acción en cada episodio depende sólo del episodio en sí mismo.
* El agente no necesita pensar con tiempo.

Entorno secuencial

* La decisión presente del agente puede afectar a decisiones futuras.
* Las acciones que se realizan a corto plazo pueden tener consecuencias a largo plazo.
* Los medios episódicos son más simples que los secuenciales porque la gente no necesita pensar con tiempo.
* Ejemplo: el ajedrez y el taxista.
* **Estático vs dinámico vs semidinámico**

Dinámico

* El entorno puede cambiar cuando el agente está deliberando.
* Los medios dinámicos están preguntando continuamente al agente qué quiere hacer; si no se ha decidido aún, entonces se entiende que ha tomado la decisión de no hacer nada. Ej.: taxista.

Estático

* El entorno no cambia cuando el agente está deliberando.
* Son fáciles de tratar ya que el agente no necesita estar pendiente del mundo mientras está tomando una decisión sobre una acción, ni necesita preocuparse sobre el paso del tiempo. Ej.: crucigramas.

Semidinámico: El entorno no cambia con el paso del tiempo, pero el rendimiento del agente sí. Ej.: el ajedrez cuando se juega con un reloj.

* **Discreto vs continuo**
* La distinción entre discreto y continuo se puede aplicar al estado del medio, a la forma en la que se maneja el tiempo y a las percepciones y acciones del agente.
* Ejemplo con variables discretas: El ajedrez tiene un medio con estados discretos y tiene un número finito de estados distintos. Además, tiene un conjunto discreto de percepciones y acciones.
* Ejemplo con variables continuas: El taxista conduciendo define un estado continuo y un problema de tiempo continuo: la velocidad y la ubicación del taxi y de los otros vehículos pasan por un rango de valores continuos de forma suave a lo largo del tiempo. La conducción del taxista es también continua (ángulo de dirección, etc.).
* Ejemplo ambas: Las imágenes captadas por cámaras digitales son discretas, en sentido estricto, pero se tratan típicamente como representaciones continuas de localizaciones e intensidades variables.
* **Conocido vs desconocido**

Conocido:Se conocen las reglas que rigen en el ambiente.

**Estructura de los agentes**

El trabajo de la IA es implementar programas de agente que implementen la función del agente. Se asume que este programa se ejecutará en algún tipo de computador con sensores físicos y actuadores, lo cual se conoce como arquitectura.

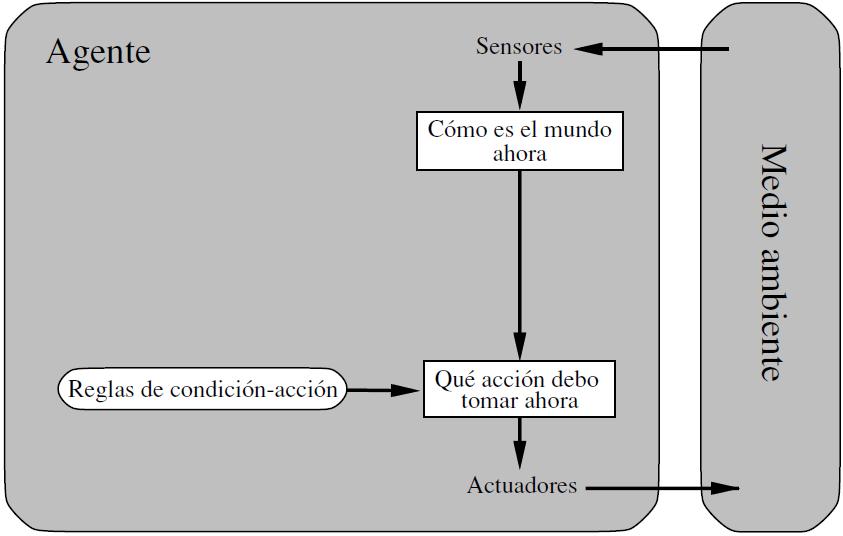
Los programas de los agentes reciben las percepciones actuales como entradas de los sensores y devuelven una acción

a los actuadores. Distinto a la función del agente que recibe la percepción histórica completa.

Hay cuatro tipos básicos de programas para agentes que encarnan los principios que subyacen en casi todos los sistemas inteligentes: reactivos simples, reactivos basados en modelos, basados en objetivos y basados en utilidad.

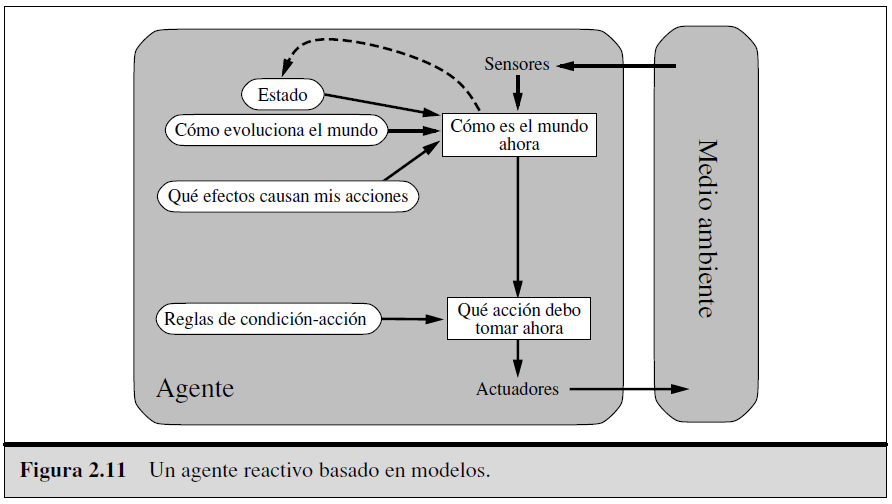
**Agente reactivo o reflejo simple**

* El más sencillo pero la inteligencia es muy limitada. Toman decisiones en base a la percepción actual. Los programas siguen la regla *condición-acción* (if-then).
* Muy fáciles de programar, pero por cada posible condición se tiene que establecer qué acción tomar.
* Los bucles infinitos son a menudo inevitables para este tipo de agentes que operan en algunos entornos parcialmente observables. Salir de los bucles es posible si los agentes pueden seleccionar sus acciones aleatoriamente (esto puede mejorar los resultados que proporciona un agente reactivo simple determinista). En entornos de agentes individuales, el comportamiento aleatorio no es normalmente racional.
* Actúa de acuerdo a la regla cuya condición coincida con el estado actual, definido por la percepción.



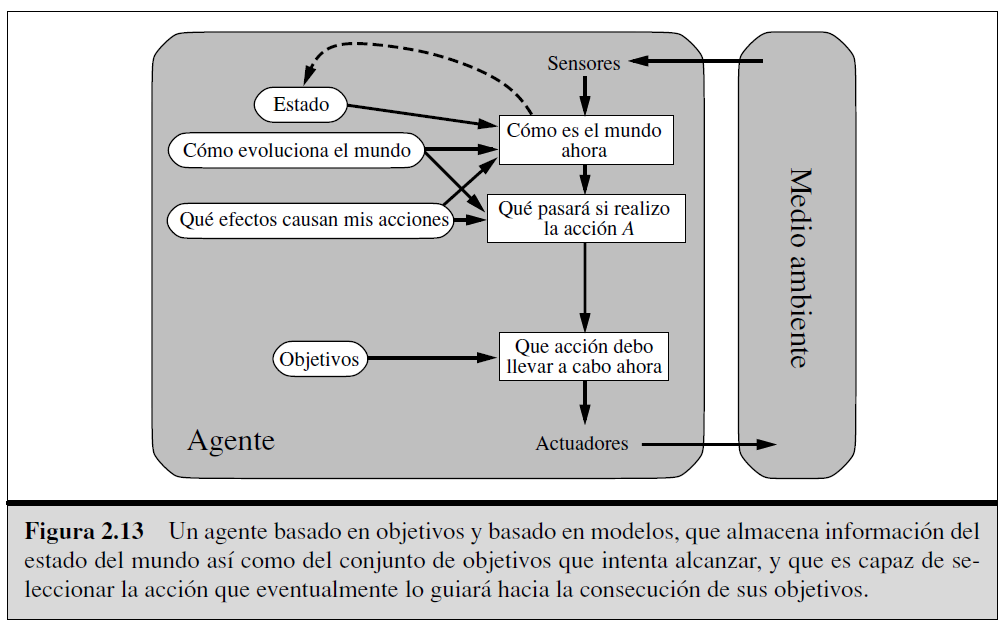
**Agente reflejo basados en modelos**

* Mantienen un estado interno que depende de la historia de percepciones. Es la forma más efectiva que tienen los agentes de manejar la visibilidad parcial es almacenar información de los pares del mundo que no pueden ver*.*
* Hay 2 tipos de conocimientos para actualizar la información del estado interno: cómo evoluciona el mundo independientemente del agente y cómo afectan al mundo las acciones del agente. Este conocimiento acerca de cómo funciona el mundo se denomina **modelo**del mundo.



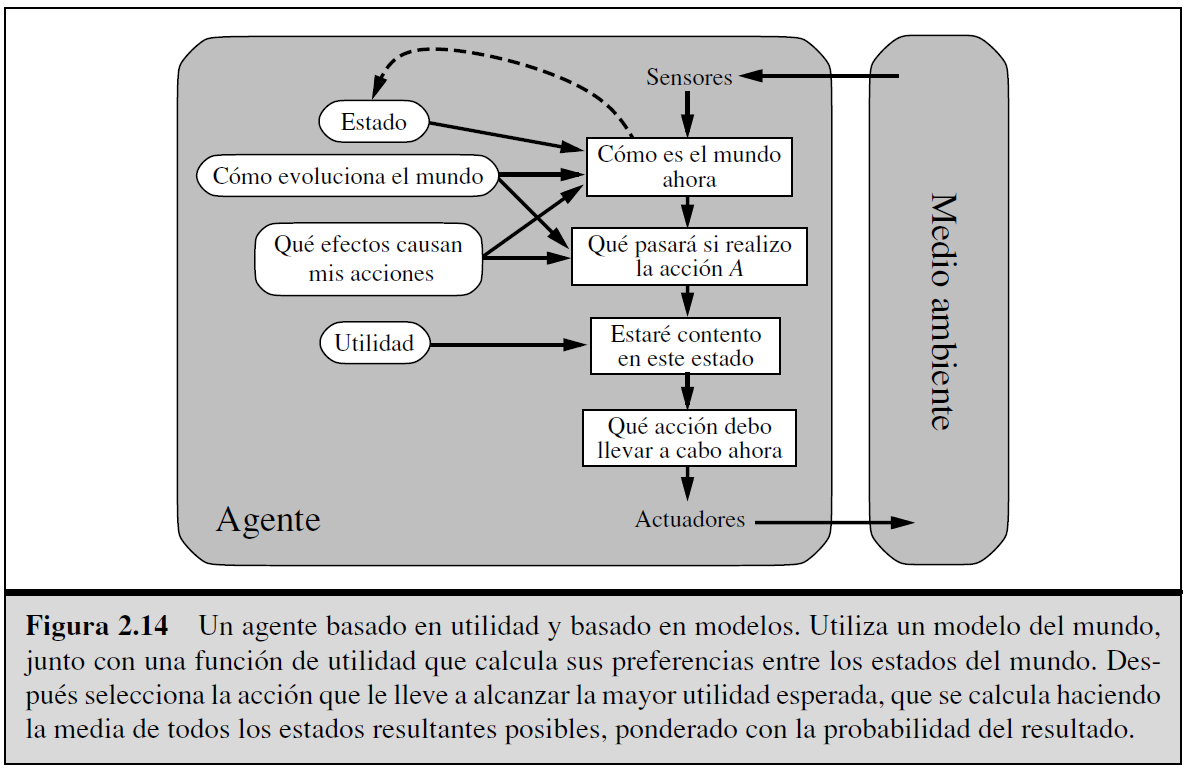
**Agente basado en objetivos o metas**

* El conocimiento sobre el estado actual del mundo no es siempre suficiente para decidir qué hacer. Además de la descripción del estado actual, el agente necesita información sobre su meta que describa las situaciones que son deseables.
* El programa del agente se puede combinar con información sobre los resultados de las acciones posibles para elegir las acciones que permitan alcanzar el objetivo. Entonces, los distintos estados se clasifican en **meta** o **no meta**.
* Son más flexibles ya que el conocimiento que soporta su decisión está representado explícitamente y puede modificarse.
* La toma de decisiones de este tipo es diferente de las reglas de condición–acción.



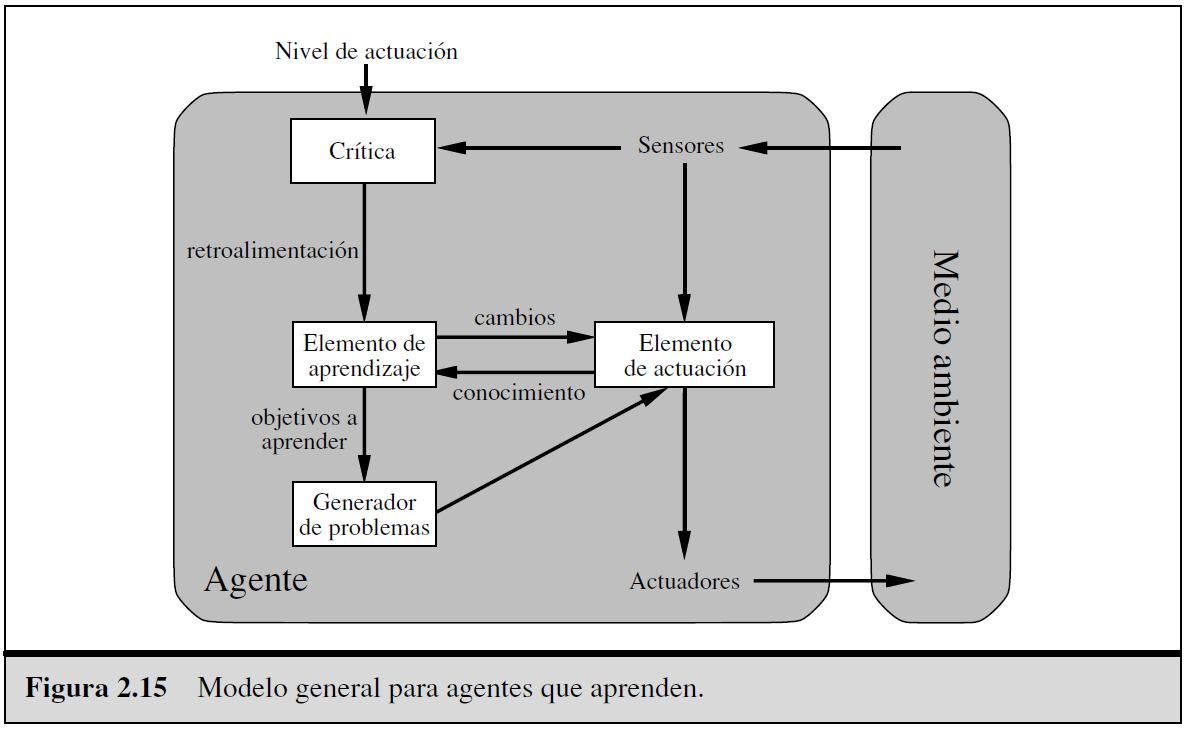
**Agente basado en utilidades**

* Se asigna una utilidad a cada estado. La medida de utilidad indica qué tan deseable es estar en ese estado.
* Un agente racional maximiza la utilidad esperada.
* La función de utilidad es una internalización de la medida de performance. Proyecta un estado (o una secuencia de estados) en un número real que representa un nivel de felicidad.
* Un agente que posea una función de utilidad explícita puede tomar decisiones racionales, y lo puede hacer con la ayuda de un algoritmo de propósito general que no dependa de la función específica de utilidad a maximizar. De esta forma, la definición global de racionalidad se transforma en una restricción local en el diseño de agentes racionales que se puede expresar con un simple programa.



**Agentes que aprenden**

* Son autónomos.
* Exploran cosas que no conocen.
* Tienen 4 componentes:
* Elemento de aprendizaje: realiza mejoras.
* Elemento de actuación: selecciona acciones externas, recibe estímulos y determina acciones.
* Criticas: realimenta al elemento de aprendizaje, determina cómo se debe modificar el elemento de actuación para proporcionar mejores resultados en el futuro.
* Generador de problemas: sugiere acciones que lo guiarán hacia experiencias nuevas e informativas.



**Agentes que resuelven problemas**

Es una clase de agende basado en objetivos que deciden qué hacer para encontrar secuencias de acciones que conduzcan a los estados deseables.

Pasos para solucionar un problema:

1. Formulación del objetivo o meta: decidir qué estados son objetivo, basado en la situación actual y la medida de rendimiento del agente.
2. Formulación del problema: decidir qué acciones y estados se van a considerar.
3. Buscar una solución: examinar posibles acciones y armar una secuencia.
4. Ejecutar la solución: ejecutar la secuencia de acciones.

Un **problema** puede ser definido formalmente mediante 5 componentes:

* Un **estado inicial.**
* Unconjunto de **acciones** disponibles para el agente en cada momento.
* Un **modelo de transición** o **función sucesor** que devuelve el estado resultante de aplicar la acción.
* Un **test objetivo** o **comprobación de meta**.
* Un **Costo de camino** o **costo de paso** que asigna un costo numérico a cada acción. La función costo refleja la medida de rendimiento.

Una **solución** es una secuencia de acciones que, aplicadas al estado inicial, me dejan en un estado meta.

Una **solución óptima** es una solución donde el **costo** de ruta es el **mínimo** de todas las soluciones posibles.

Formulando problemas

Para formular los problemas es obligatorio hacer **abstracciones**, es decir, eliminar todas aquellas cuestiones que son irrelevantes.

* **Abstracción de Estados:** conservar los mínimos detalles necesarios para cumplir el objetivo.
* **Abstracción de Acciones:** contemplar solo las acciones necesarias, expresadas de la manera más sencilla posible.

Buscando soluciones

Los problemas se resuelven mediante búsqueda a través del espacio de estados.

Las técnicas de búsqueda que utilizan un **árbol de búsqueda** explícito generado por el estado inicial y la función sucesor, definiendo así el espacio de estados.

Las aristas del árbol son acciones y los nodos se relacionan a estados.

El primer paso es comprobar si éste es un estado objetivo. Si no estamos en un estado objetivo, hay que considerar otros estados **expandiendo** el estado actual; es decir aplicando la función sucesor al estado actual y **generar** así un nuevo conjunto de estados.

Continuamos escogiendo, comprobando y expandiendo hasta que se encuentra una solución o no existan más estados para expandir. El estado a expandir está determinado por la **estrategia de búsqueda**.

La **frontera** o **lista abierta** es la lista de nodos hojaque se han generado y que todavía no fueron expandidos o procesados.

El **conjunto de nodos explorados** o **lista cerrada** son todos aquellos estados que fueron procesados.

La **búsqueda en grafo** tiene en cuenta la lista cerrada; mientras que la **búsqueda en árbol** no.

Infraestructura para algoritmos de búsqueda

La estructura de un nodo se define mediante un estado, un nodo padre, una acción y un costo de camino (desde el estado inicial al nodo, indicado por los punteros a los padres)

La estructura para la frontera es una lista que se comporta como cola, pila o cola priorizada. El conjunto de estados explorados puede ser implementado como una tabla hash.

Los nodos y estados no son lo mismo. Un nodo es una estructura de datos y un estado es una configuración del mundo. Los nodos están en caminos particulares pero los estados no.

**Midiendo performance de los algoritmos**

* Completitud: el algoritmo asegura encontrar una solución, si la hay.
* Optimalidad: el algoritmo encuentra una solución óptima.
* Complejidad temporal: ¿Cuánto tiempo toma encontrar la solución?
* Complejidad espacial: ¿Cuánta memoria necesito para encontrar la solución?

Las medidas de complejidad se van a expresar en términos de:

* 🡪 factor de ramificación o máximo número de sucesores de cualquier nodo
* 🡪 menor profundidad de algún nodo meta.
* 🡪 longitud máxima de camino en el grafo de estados.

El tiempo se mide en términos de la cantidad de nodos generados; mientras que el espacio según la cantidad de nodos en memoria.

Para valorar la eficacia de un algoritmo de búsqueda, se puede considerar el costo de la búsqueda o podemos utilizar el coste total, que combina el costo de la búsqueda y el costo del camino solución encontrada. Para nuestros problemas utilizaremos el costo de búsqueda.

**Estrategias de búsqueda sin información**

Estas estrategias son llamadas de este modo porque no tienen información adicional acerca de los estados más allá de la que proporciona la definición del problema. Solo puede generar sucesores y distinguir un estado objetivo de uno que no lo es. Se diferencian por el orden de expansión de los nodos.

**Búsqueda en amplitud**

Se expanden todos los nodos a una profundidad en el árbol antes de expandir cualquier nodo del siguiente nivel.

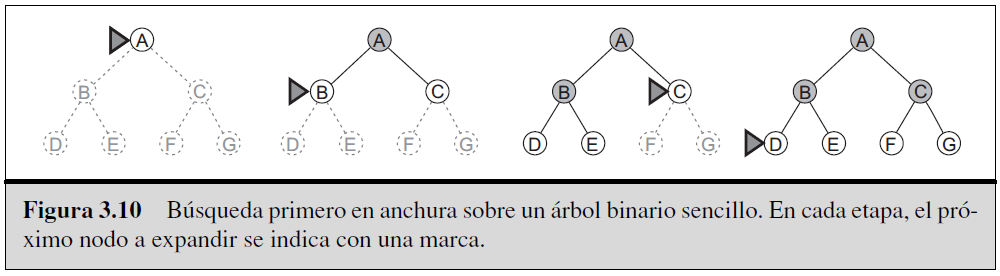
Utiliza una cola en la frontera con comportamiento FIFO.

Es **completa**: va a encontrar la solución si la hay (si hay varias, la más superficial).

**No siempre es** **óptima** porque recorre por niveles y puede que la primera solución que aparezca no sea la más óptima. Es óptima cuando todas las acciones tienen igual costo.

Utiliza gran cantidad de tiempo y memoria, ya que mantiene todos los nodos recorridos y expandidos.

En memoria hay nodos.



Modificación en el código: Cuando se expande un nodo ya se pregunta si es meta con los expandidos para no expandir los siguientes nodos que hay en la frontera.

**Búsqueda de Costo uniforme**

Utiliza una **cola priorizada** en la frontera, ordenada por el costo acumulado del camino. La función g(n) devuelve el costo del camino desde el nodo inicial al nodo actual. Si todos los costos son iguales, funciona igual que la búsqueda en amplitud.

Dos diferencias en el algoritmo:

* la comprobación de meta se realiza recién cuando se va a expandir el nodo.
* se agrega un chequeo por si se encontró un estado ya generado, pero de menor costo.

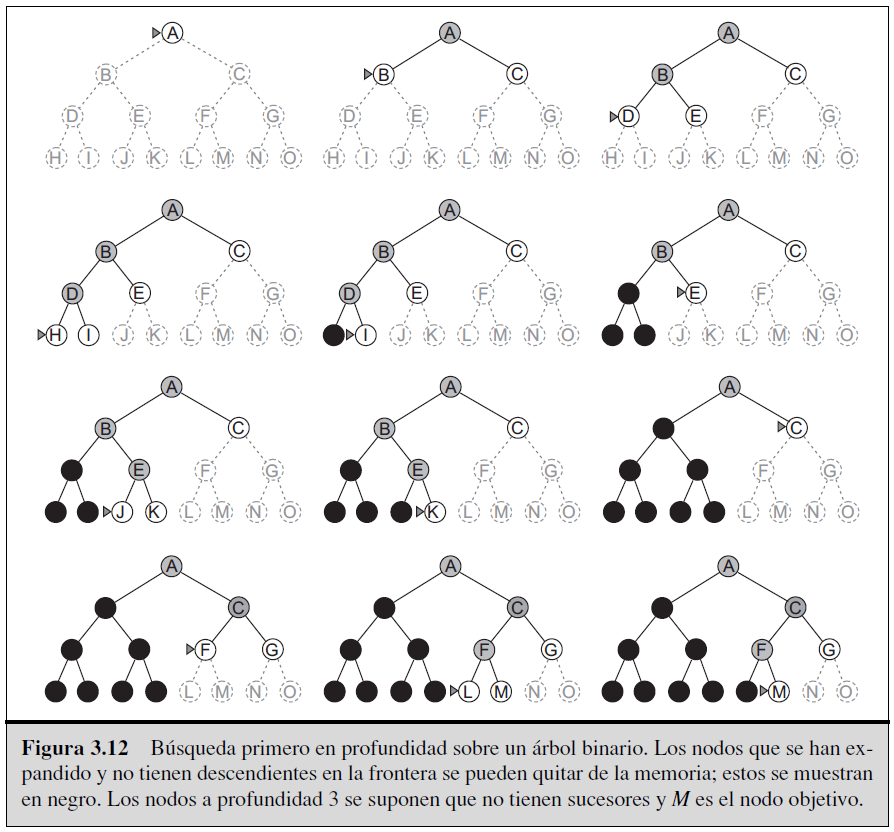
En búsqueda en grafo: Si al expandir un nodo encuentra un nodo que ya procesó o está en cola, se fija en la función y si el camino actual es más corto, lo reemplaza.

Es **completa** si el costo de cada paso es mayor o igual a alguna constante positiva pequeña .

Encuentra la **solución óptima**, ya que expande nodos con costo creciente (la primera solución encontrada será la óptima).

No se puede estimar fácilmente la performance, ya que depende de los costos de los caminos más que de las profundidades. La complejidad en tiempo y espacio del peor caso del algoritmo es ( es el costo de la solución óptima), la cual puede ser mucho mayor que .

**Búsqueda en profundidad**

Esta estrategia siempre expande el nodo más profundo en la frontera actual del árbol de búsqueda. La búsqueda procede inmediatamente al nivel más profundo del árbol de búsqueda, donde los nodos no tienen ningún sucesor. Cuando esos nodos se expanden, son quitados de la frontera, así entonces la búsqueda «retrocede» al siguiente nodo más superficial que todavía tenga sucesores inexplorados.

Utiliza una **pila** (Cola LIFO) en la frontera.

* La **completitud** depende de si hacemos búsqueda en árbol o en grafo. En árbol puede que quede en un bucle infinito, dependiendo del problema.
* **No es óptima** porque la primera solución que encuentre puede que no sea la mejor.
* La complejidad temporal es (puede ser peor que )
* La complejidad espacial es . Se pueden borrar nodos de memoria.

La variante backtracking search utiliza aún menos memoria.

**Búsqueda en profundidad limitada**

Igual a la búsqueda en profundidad, con la salvedad que se limita la profundidad máxima a alcanzar. No siempre es posible determinar L. (límite establecido).

* Introduce otro problema para la completitud, el límite puede ser inferior a la profundidad de la mejor solución.
* Su complejidad en tiempo es .
* Su complejidad en espacio es .

**Búsqueda en profundidad iterativa**

Encuentra el mejor límite de profundidad aumentándolo gradualmente si no encuentra solución.

Cada vez que aumenta de nivel, empieza a recorrer de cero por lo que tarda mucho tiempo.

**Búsqueda bidireccional**

Ejecuta 2 búsquedas simultáneas: una hacia adelante desde el estado inicial y otra hacia atrás desde el estado meta. En algún momento se van a cruzar los caminos por lo que se ahorra recorrer nodos innecesarios.

* hay que tener acciones reversibles
* hay que conocer el estado meta

Es **completa** si una de las búsquedas es completa.

Las dos tiene que ser **óptimas**.

Comparación de las estrategias de búsqueda no informada

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Criterio** | **En amplitud** | **Costo uniforme** | **Profundidad** | **Profundidad limitada** | **Profundidad iterativa** | **Bidireccional** |
| **Completa** |  |  |  |  |  |  |
| **Tiempo** |  |  |  |  |  |  |
| **Espacio** |  |  |  |  |  |  |
| **Óptima** |  |  |  |  |  |  |
| a: completa si b es finita. b: completa si los costos son . c: óptima si los costos son iguales. d: si en ambas direcciones se utiliza la búsqueda en amplitud. | | | | | | |

**Estrategias de búsqueda informada**

* Utilizan conocimiento específico del problema más allá de la definición del problema en sí mismo. Puede encontrar soluciones de una manera más eficiente que una estrategia no informada.
* best-first search utiliza una función f(n) para estimar costos. (Búsqueda de costo uniforme)
* h(n) es una función heurística que generalmente forma parte de f(n). Dado un estado, dice cuánto supone que le falta para llegar a la meta.
* h(n) no tiene en cuenta el costo hasta el momento, solamente estima desde el nodo actual cuál es el costo hasta la meta.
* h(n) = 0 <==> n es un nodo meta

Vamos a ver dos estrategias de búsqueda:

* Greedy best-first search (búsqueda avara)
* A\*

**Búsqueda avara**

Trata de expandir el nodo más cercano al objetivo, alegando que probablemente conduzca rápidamente a una solución. Así, evalúa los nodos utilizando solamente la función heurística: .

Problemas:

* Si usamos búsqueda en árbol puede ser incompleta; si usamos búsqueda en grafo es completa en espacios finitos.
* No es óptima.
* La complejidad temporal y espacial son complicadas de calcular. En el peor de los casos es ; pero esto depende mucho de la heurística.
* Usa el mismo algoritmo que la búsqueda de costo uniforme, reemplazando por .

**Búsqueda A\* (A estrella)**

Minimiza el costo total estimado de la solución

El camino completo cuesta: . Es una función creciente.

Usa el mismo algoritmo que la búsqueda de costo uniforme, reemplazando por

Los costos de las acciones no pueden ser menores a 0.

Heurística y costo deben estar en la misma unidad de medida.

Si satisface determinadas condiciones, A\* es completa y óptima:

* Admisibilidad: Nunca sobreestima el costo de llegar a la meta. Puede estimar de menos pero no de más. Generalmente asegura la consistencia.
* Consistencia (necesaria solo para búsqueda en grafo):

Optimalidad de A\*

A\* busca en contornos de nodos del mismo costo total.

Si C\* es el costo de la solución óptima:

* A\* puede expandir algunos nodos dentro del contorno de la solución
* A\* expande todos los nodos con f(n) < C\*

**Parcial: A\* es óptimamente eficiente para cualquier heurística: no hay otro algoritmo que garantice expandir menos nodos que A\*; asegurando la optimalidad de la solución. No hay otro algoritmo que asegure encontrar una solución óptima y que expanda menos nodos que A\*.**

Puede que haya otros algoritmos que encuentren la misma solución en menos pasos, pero no garantizan la optimalidad como A\*.

Es el mejor algoritmo según optimalidad eficiente.

**Funciones heurísticas**

Una manera de caracterizar la calidad de una heurística es a través del factor de ramificación efectivo b\*

Una heurística bien diseñada tiene b\* cercano a 1.

Si para cada nodo n, h2(n) >= h1(n) decimos que h2 domina a h1.

**Generando heurísticas**

* Desde problemas relajados: el costo de una solución óptima de un problema relajado es una heurística admisible para el problema original. Un problema relajado es similar al original, pero con menos restricciones.
* Si se encuentran varias heurísticas alternativas, podemos generar una que domine a todas de la siguiente manera:
* Desde subproblemas: el costo de solucionar un subproblema es menor que el de solucionar el problema completo

En las **bases de patrones** se almacenan los costos exactos para varios subproblemas, costos que luego son usados para calcular las heurísticas. Calcular los costos puede llevar tiempo, pero la búsqueda sería siempre rápida. El problema es el espacio que ocupan los datos.

**Más allá de la búsqueda clásica – Algoritmos de optimización**

Vamos a relajar las restricciones hechas previamente, donde los ambientes tenían que ser observables, determinísticos y conocidos; y las soluciones eran secuencias de acciones.

**Búsqueda local**

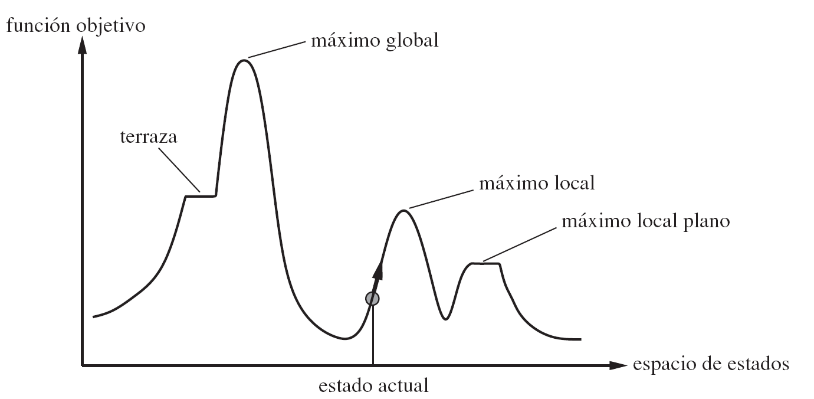
Se usan cuando no importa el camino para llegar a la solución. Ejemplo: 8 reinas.

Mantienen solo un nodo en memoria y se mueven a sus vecinos (usan poca memoria). El problema es que se puede repetir un estado.

Son útiles en problemas de optimización.

Generalmente usan una formulación de estado completa (por ejemplo, en el caso de las 8 reinas, usa un tablero con las 8 reinas ya colocadas en alguna posición random).

Una **función objetivo** determina qué tan bueno es un estado. Esta función la podemos dibujar respecto al espacio de estados para obtener un paisaje del espacio de búsqueda. Los estados están ordenados en cuanto a nivel de cercanía (cantidad de acciones).



La función objetivo reemplaza al es\_meta porque importa qué tan útil es el estado. Ejemplo: en las 8 reinas, la cantidad de reinas atacadas puede ser la función y la meta va a ser el valor que dé menos cantidad de reinas atacadas.

Un algoritmo es completo si siempre encuentra una solución, es óptimo si encuentra un máximo o mínimo global.

Todos los algoritmos de búsqueda local pueden no encontrar la mejor solución.

**Algoritmos**

* **Ascenso de colina (Hill climbing)**

Es un bucle que siempre se mueve hacia arriba y termina cuando llega a un pico.

Tiene problemas con máximos locales, crestas (secuencias de máximos locales, tipo escalera) y mesetas.

No es completo porque puede no encontrar la mejor solución. Tampoco es óptimo porque puede no encontrar el máximo global.

Ventajas: es muy rápido y no ocupa memoria, prácticamente.

Variantes con random:

* **Stochastic hill climbing**: elige al azar entre los movimientos ascendentes vecinos.
* **First-choice hill climbing**: genera los sucesores aleatoriamente de a uno, hasta que uno sea mejor que el actual (es útil cuando el factor de ramificación es alto).
* **Random restart hill climbing**: ejecuta varias iteraciones de hill climbing tradicional, comenzando desde distintos estados iniciales generados aleatoriamente.
* **Temple simulado (Simulated annealing)**

Combina el ascenso rápido con la aleatoriedad.

A medida que transcurre el tiempo (o disminuye la temperatura) la posibilidad de elegir un sucesor peor disminuye.

En vez de elegir el mejor movimiento, elige un movimiento aleatorio. Si mejora la situación, es siempre aceptado. Si es peor, según la probabilidad actual, se lo queda o no. La probabilidad depende de cuán peor es y de la cantidad de tiempo transcurrido (temperatura).

* **Búsqueda de Haz local**

Mantiene k nodos en memoria, en principio generados aleatoriamente. Por cada paso se generan todos los sucesores, se eligen lo k mejores y se repite (o, si alguno es objetivo, se elige ese y para el algoritmo).

La información importante es compartida entre las distintas búsquedas paralelas. El algoritmo rápidamente abandona las búsquedas infructuosas y mueve sus recursos a donde se hace la mayor parte del progreso.

Los k estados pueden concentrarse rápidamente en un sector pequeño del espacio de estados. Puede que el sector sea malo y no logre encontrar el máximo global.

Aparece una variante llamada **stochastic beam search**: elige a k sucesores aleatoriamente, con la probabilidad de elegir a un sucesor como una función creciente de su valor (a mayor valor, mayor probabilidad de seleccionarlo).

* **Algoritmos genéticos**

Es una variante de la búsqueda de haz estocástica, solo que los sucesores se generan combinando dos estados padres.

Comienza con un conjunto llamado la población, compuesto de k estados o individuos. Cada individuo está representado como una cadena sobre un alfabeto finito (el más común, una cadena de 0s y 1s).

Se usa una función fitness u objetivo, la cual devuelve valores mayores cuanto mejor sea el individuo.

Se eligen pares de individuos para ser reproducidos. La elección es al azar, con probabilidades proporcionales al fitness de cada individuo

Los pares son sometidos a una operación de crossover (se elige un punto de cruce y se mezclan los individuos, generando dos con una parte de cada uno).

Finalmente se pueden realizar mutaciones en los individuos resultantes con una pequeña probabilidad (se modifica algún dígito, por ejemplo).

**Más búsqueda...**

* **Búsqueda con acciones no deterministas**: las soluciones devuelven planes de contingencia en lugar de secuencias de acciones. (si la acción es moverse para adelante y se mueve para la derecha, el plan le dice que se mueva a la izquierda, y así). Modifica el algoritmo.
* **Búsqueda en ambientes parcialmente observables**: los algoritmos trabajan con estados de creencia en lugar de estados reales.
* **Búsqueda online**: cuando el mundo o las acciones no son conocidos se necesita intercalar la búsqueda con la ejecución. Después de cada acción, un agente online recibe una percepción que indica qué estado ha alcanzado; de esta información, puede aumentar su mapa del entorno. El mapa actual se usa para decidir dónde ir después. Esta intercalación de planificación y acción significa que los algoritmos de búsqueda online son bastante diferentes de los algoritmos de búsqueda offline vistos anteriormente.

**Problemas de satisfacción de restricciones**

Un problema es considerado de **satisfacción de restricciones** cuando se lo representa con un conjunto de variables a las cuales se le tienen que asignar valores de manera de no violar las restricciones.

No consumen memoria.

Definiendo problemas de satisfacción de restricciones

Poseen 3 componentes: *X*, *D* y *C*

* **X**: conjunto de **variables**, {x1, x2, ..., xn}

Cada variable tiene un conjunto de **dominios**:

* + **D**: conjunto de dominios, {d1, d2, ..., dn}.

Cada dominio posee un conjunto de valores permitidos para la variable relacionada.

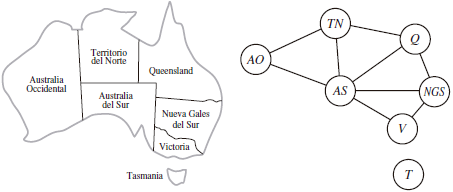
* **C**: conjunto de **restricciones** que especifica las combinaciones permitidas de valores.

Una restricción consiste en una tupla *<scope, rel>*, donde *scope* es una lista de las variables afectadas y *rel* es la condición que se debe cumplir.

Cada estado en un CSP está definido por una asignación de valores a algunas o todas las variables.

* **Asignación consistente:** no viola ninguna de las restricciones.
* **Asignación completa:** todas las variables tienen algún valor asignado.
* Una **solución** es una asignación **completa** y **consistente**.

Es bueno visualizar un CSP como un grafo de restricciones. Los nodos son las variables y los arcos las restricciones.



Ventajas de CSP sobre búsqueda tradicional

* Posee una representación natural para una gran variedad de problemas.
* Son más rápidos ya que pueden eliminar grandes partes del árbol de búsqueda limitando el dominio.
* En las búsquedas tradicionales solo podemos saber si un nodo es solución o no. En CSP podemos saber si una asignación parcial noconduce a una solución y porquéuna asignación no es solución.
* En CSP hay heurísticas genéricas para este tipo de problemas.

Variantes del formalismo de CSP

* Puede haber dominios **discretos y finitos**, discretos e infinitos, continuos e infinitos.
* Tipos de restricciones:
  + **Restricciones unarias:** restringen el valor de una variable.
  + **Restricciones binarias:** la restricción relaciona dos variables. Si un CSP solo posee este tipo de restricciones es un CSP binario.
  + **Restricciones globales:** involucran un número arbitrariode variables. Uno de los ejemplos más conocidos es *AllDiff*.
    - Pueden ser representadas mediante el uso de hiper-grafos.
    - Cualquier CSP con restricciones globales puede ser convertido a un CSP binario, aunque una restricción global puede ser más sencilla de leer y es posible diseñar inferencias de propósito especial que funcionen con las restricciones globales.
    - Los algoritmos pueden quedar estancados.
    - Las restricciones no se pueden chequear hasta que no estén todas las variables asignadas. Es mejor usar restricciones binarias para controlar antes las restricciones.
    - Son comunes en problemas reales y pueden ser manejadas por algoritmos especiales.
    - Alldiff: comprueba que todos los valores sean distintos. Si len(set(variables)) > len(set(dominios)) => Inconsistencia
    - Atmost: comprueba que como mucho se usen X recursos.

**Propagación de restricciones**

La propagación de restricciones es un tipo de inferencia en la que se reduce el número de valores legales que puede tomar una variable. Puede realizarse como una tarea de preprocesamiento (se verifican las restricciones antes de buscar) o intercalada con la búsqueda.

Distintos tipos de **consistencia local**:

* Consistencia de nodo:

Una variable es nodo-consistente si todoslos valores de su dominio satisfacen las restricciones unarias.

Una red es nodo-consistente si todas sus variables son nodo-consistentes.

Ej:SAposee el dominio *{ red, blue, green }* pero hay una restricción de que a *SA* no se le puede asignar *green*. La inferencia hace que *green* sea eliminado del dominio de *SA*.

* Consistencia de arco:

es arco-consistente respecto a si para cada valor de existe algún valor en que satisfaga la restricción binaria en el arco.

Una red es arco-consistente si cada variable es arco-consistente con las demás variables.

El algoritmo más popular para asegurar consistencia de arco se denomina **AC-3**. La complejidad es *O(cd^3)*

Ej:se tienen dos variables, *X* e *Y*, donde los dominios de cada una son los dígitos. Se posee además esta restricción binaria: *.*

Para hacer que X sea arco-consistente con Y hay que reducir el dominio de X a {0,1,2,3}; nótese que 4^2 = 16, valor que no pertenece al dominio de Y.

Para hacer Y arco-consistente con X hay que reducir el dominio de Y a {0,1,4,9}

* Consistencia de camino:

Un conjunto de 2 variableses camino-consistente con respecto a una tercer variable si, para cada asignación consistente con las restricciones de, existe una asignación a que satisface las restricciones en y .

* K-Consistencia:

Un CSP es k-consistente si, para cualquier conjunto *k - 1* de variables y para cualquier asignación consistente de esas variables, un valor consistente siempre puede ser asignado a cualquier *k-esima* variable.

A mayor valor de k, menor el tiempo de búsqueda de la solución; pero, la cantidad de tiempo y memoria utilizado para propagar la consistencia crece de manera exponencial a k.

Frecuentemente se utiliza consistencia de arco; menos frecuentemente, consistencia de camino.

**Algoritmos de CSP**

**Backtraking search for CSPs**

* Trabaja con asignaciones incompletas.
* Elige valores para una variable a la vez y vuelve atrás cuando una variable no tiene ningún valor legal para asignar.
* **Conmutatividad** 🡪 el orden de aplicación de cualquier conjunto de acciones no tiene ningún efecto sobre el resultado. Asignando valores a variables, alcanzamos la misma asignación parcial sin tener en cuenta el orden. Por lo tanto, todos los algoritmos de búsqueda para el CSP generan los sucesores considerando asignaciones posibles para sólo una variable en cada nodo del árbol de búsqueda.
* No se especifica estado inicial, action, result o goal test (son siempre los mismos)
* La performance se puede mejorar mediante heurísticas genéricas.

Formas de asignar variables:

* Selección de variable:
* La siguiente.
* Minimun remaining values (MRV):elige la variable más restringida primero (tamaño del dominio). Achica el árbol, es más rápido.
* Grado heurístico:variable con mayores apariciones en las restricciones. Se usa generalmente para desempatar luego de aplicar MRV.
* Orden de valores del dominio:
  + Least constraining value (LCV): elige primero el valor que menos opciones quita al resto de las variables.

Intercalando búsqueda e inferencia

* **Forward checking:** cuando una variable *X* es asignada, chequea toda variable conectada eliminando de sus dominios valores inconsistentes con la asignación de *X*.
  + Se complementa bien con MRV ya que elimina valores y permite discernir las variables más restringidas.
* **MAC (Maintaining Arc Consistency)**: llama al algoritmo AC3 luego de cada asignación para aplicar consistencia de arco recursivamente.

**Local Search for CSPs**

* Se empieza con asignaciones completas.
* En cada paso se elige al azar una variable que esté involucrada en una restricción violada y se cambia el valor.
* Para elegir el valor de una variable se usa la heurística de mínimos conflictos**.** (elige el valor que menos restricciones viola)
* Se puede utilizar para mejorar soluciones cuando se producen cambios en el problema. Ej. problemas de planeación.