Notas del Curso:
Inteligencia Artificial

Facultad de Ciencias de la Computación
Benemérita Universidad Autónoma de Puebla

M.I.A. José Juan Palacios

Otoño 2002

Estas notas de curso presentan una síntesis de los temas a tratar en el curso. La bibliografía principal es:

Artificial Intelligence, a modern approach
Stuart Russell and Peter Norvig
Prentice Hall

por lo que se recomienda que se adquiera dicho texto (hay una versión muy bien redactada en español).

El propósito del curso es presentar un panorama general de las técnicas que se han utilizado y se utilizan en el área multidisciplinaria conocida como Inteligencia Artificial desde la perspectiva de las ciencias de la computación, con especial atención al aspecto de *interacción humano -computadora*. Sin embargo, dado su carácter multidisciplinario, resulta importante señalar al menos los retos y problemas existentes, las limitaciones y perspectivas a futuro.

El enfoque que se persigue es presentar los tópicos de forma gradual (desde las técnicas sencillas hasta las más elaboradas) bajo la noción de *Agente Inteligente*. Esencialmente, un agente es una entidad independiente que percibe su entorno mediante sensores y que actúa (afecta al entorno, al menos parcialmente) mediante efectores de acuerdo a sus propósitos ó *metas*, las cuales pueden ser tan complejas y elaboradas ó tan primitivas como asegurar -incluso por instinto- su propia supervivencia. Sin duda, el aspecto más importante en la concepción, estudio e implantación de agentes inteligentes se puede plantear como la pregunta: *¿cómo hacer lo correcto?*. Para responder a tal cuestionamiento, a lo largo del curso se revisan las técnicas clásicas, como son:

1. Resolución de problemas mediante técnicas de Búsqueda
   (a) Búsqueda sin información adicional: Primero en Amplitud, Primero en Profundidad, Búsqueda de Costo Uniforme.
   (b) Búsqueda con información: Primero el Mejor, Búsqueda Ávida, Aplicación de Heurísticas: A^*.
   (c) Algoritmos de Mejora Iterativa: *Ascenso de Colina, Recocido Simulado.*
2. Técnicas de Juegos: los juegos entre dos agentes antagónicos como problema de búsqueda
   (a) Funciones de Evaluación
   (b) Poda Alfa-Beta
3. *Representación del conocimiento e inferencia mediante lógica simbólica*
4. *Planificación,*
5. *Aprendizaje automático y procesamiento del lenguaje natural.*

Todas las técnicas se concretan en la implantación de módulos (software) los cuales incorporados en los agentes les ofrecen cierto grado de capacidad.

No es coincidencia que la perspectiva (o paradigma) de *orientación a agentes* para el análisis y programación de sistemas sea un área muy rica de investigación. Desde sistemas distribuidos hasta interacción humano computadora, los agentes de software juegan un papel protagonista en el diseño y construcción de sistemas modernos. En gran medida se puede considerar a un agente computacional como la extensión natural del paradigma orientado

a objetos ya que éste último ofrece un sustento sólido para las necesidades en sistemas orientados a agentes. La interacción de los agentes con su entorno se fundamenta en la *comunicación*, la cual puede implantarse uniformemente mediante *paso de mensajes*.

Además, tradicionalmente en la Inteligencia Artificial computacional se han utilizado lenguajes de programación simbólicos como *Lisp, Prolog, ML*, etc. los cuales tienen la ventaja de permitir describir de manera declarativa los programas. Aunado a ello, los programas escritos en tales lenguajes asemejan mucho las definiciones formales que se utilizan para plantear las teorías que describen el fenómeno en estudio, además de ser sencillos de aprender por estar orientados a la descripción del problema -propiedades- más que al aspecto del cómo deba de ser resuelto por la computadora.

Por ello, una *simbiosis* entre la orientación a objetos y los lenguajes de programación declarativos (especialmente Prolog) resulta ser muy atractiva. En éste curso las técnicas se implantarán en Java y Prolog; teniendo como proyecto final la implantación de un agente inteligente que actúe dentro de un ambiente antagónico.

Capítulo 1

Introducción

En su concepción integral, la Inteligencia Artificial es una ciencia multidisciplinaria fundamentalmente empírica que tuvo su origen y difusión principal dentro de las ciencias computacionales en la conferencia de verano de 1966 en Darmouth College. El nombre *Inteligencia Artificial (Artificial Intelligence)* fué propuesto por John McCarthy. Varios investigadores en diversas áreas presentaron sus trabajos en progreso de la aplicación de la computadora como laboratorio para probar sus teorías, fundamentalmente sustentadas en diversas formas de *razonamiento simbólico.*

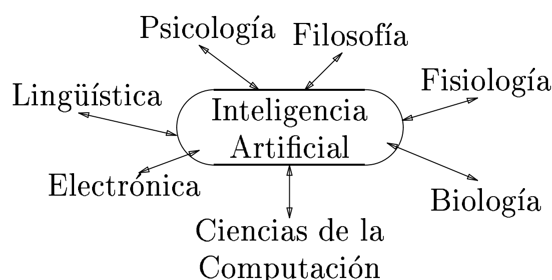


Figura 1.1: La Inteligencia Artificial como una ciencia multidisciplinaria.

En términos generales, la Inteligencia Artificial pretende *entender el funcionamiento de las entidades inteligentes* mediante la construcción de agentes (de software ó físicos -robots-) que exhiban habilidades y comportamiento juzgado como inteligente.

Aún cuando la definición misma de "inteligencia" resulta difícil de acordar, en términos generales podemos considerarla como *un conjunto estructurado de capacidades que permiten a una entidad (organismo natural ó artificial - robot, software-) resolver los problemas que le presenta el entorno en el cual está situado, anticipándose y adaptándose a las circunstancias en el transcurso del tiempo.* Convenientemente hacemos aquí una breve antología a H. Jiménez[1] que sintetiza una jerarquía de las capacidades que exhiben el comportamiento inteligente en los seres vivos:

A nivel primitivo, los organismos vivos cuentan con la capacidad del *reflejo condicionado* que les permite responder ante los estímulos; tales respuestas están provistas por el código

[1]**Héctor Jiménez Salazar**, *Panorama del curso Programación Lógica e Inteligencia Artificial* Fac. de Ciencias de la Computación, UAP. Julio 4, 2001.

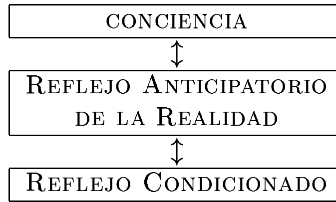


Figura 1.2: Jerarquía de capacidades principales en los organismos.

genético del organismo mismo. En el siguiente nivel, *el reflejo anticipatorio de la realidad* probablemente permite provocar la evolución en los patrones que asimila el organismo, extendiendo su memoria y su capacidad por asimilar y representar su propio mecanismo de reflejo. Tal representación dependerá de la experiencia con el medio, ya que algunas posibilidades serán más importantes que otras según representen más fielmente a la problemática del entorno. De aquí surge la necesidad de selección y almacenamiento a largo plazo de tales representaciones como formas de *operadores de pensamiento* (algunos avanzados como análisis y síntesis) para este tipo de organismo. Finalmente, la conciencia enfrenta problemas como la transmisión (no genética) del conocimiento a las nuevas generaciones, involucrando a su vez problemas de comunicación y la noción del tiempo, originados probablemente por aplicar operaciones del pensamiento a problemas que a su vez están relacionados con la esencia misma de los organismos.

El medio influye definitivamente para que el organismo desarrolle sus habilidades: la *actividad* queda en función de la *necesidad*, ya que le medio presenta problemas y el organismo busca su propia conservación (1a Ley Biológica), para lo cual, el organismo se plantea un propósito (**meta**) y gracias a la experiencia que tiene en memoria traza una forma de **plan de acción** (claro que dependiendo de la situación pueda actuar por instinto y reflejo inmediato). La experiencia del organismo se refina y acrecenta tanto por el resultado de la aplicación del plan (éxito -parcial ó total- ó fracaso) así como sus mecanismos propios de retroalimentación.

Sintetizando los puntos más sobresalientes sobre la capacidad inteligente en los seres vivos:

1. La inteligencia requiere experiencia, representada mediante formas de conocimiento (desde simplemente estímulos hasta descripciones complejas tanto de su entorno como de su propio comportamiento).

2. El mantenimiento de la inteligencia exige la prueba de planes, liberando a la inteligencia de un medio en particular.

3. El nivel de conciencia dirige la necesidad y en consecuencia, la calidad de la actividad inteligente.

La Inteligencia Artificial ha recibido varias definiciones a lo largo de su existencia, que pueden clasificarse en 4 grandes dimensiones (cada una de las cuales ha sido investigada):

Sistemas que:

piensan como humanos	piensan racionalmente
actúan como humanos	actúan racionalmente

La tabla 1.1 (tomada de Russel & Norving) presenta algunas de tales definiciones acorde a las dimensiones mencionadas:

"El esfuerzo nuevo y excitante por hacer que las computadoras piensen ... *máquinas con mentes*, en el sentido completo y literal"(Haugeland, 1985) "[La automatización de] las actividades que están asociadas al pensamiento humano, tales como la toma de decisiones, solución de problemas, aprendizaje ..." (Bellman, 1978)	"El estudio de las facultades mentales mediante el uso de modelos computacionales" (Charniak & Mc Dermott, 1985) "El estudio de los cómputos que hacen posible percibir, razonar y actuar" (Winston, 1992)
"El arte de crear máquinas que realizan funciones que requieren inteligencia cuando son llevadas a cabo por personas" (Kurzweill, 1990) "El estudio de cómo hacer que las computadoras hagan cosas que en la actualidad, las personas son mejores" (Rich & Knight, 1991)	"Un campo de estudio que busca explicar y emular el comportamiento inteligente en términos de procesos computacionales" (Schalkoff, 1990) "La rama de las ciencias de la computación que está interesada en la automatización del comportamiento inteligente" (Luger & Stubblefield, 1993)

Tabla 1.1: Algunas definiciones de Inteligencia Artificial .

Las definiciones que se encuentran en la parte superior de la tabla están enfocadas respecto los *procesos de pensamiento y razonamiento*, mientras que las situadas en la parte inferior se centran en el aspecto del *comportamiento*. Las definiciones en la columna izquierda miden el éxito en términos del *desempeño humano*, mientras que las de la derecha centran su medida de éxito en términos de un concepto *ideal* de inteligencia denominado **racionalidad**. Decimos que un sistema (Agente) es racional si hace lo correcto, alcanza sus metas dadas sus creencias seleccionando acciones que incrementan su medida de éxito.

Describiremos brevemente cada una de las cuatro categorías, las cuales han sido exploradas históricamente y han surgido tensiones entre tales enfoques centrados en el aspecto humanista vs. racional. Mientras que el enfoque hmanista debe ser una ciencia empírica, involucrando planteamiento de hipótesis y confirmación experimental, el enfoque racional involucra fundamentalmente una combinación de matemáticas e ingeniería.

Acción humana: el enfoque de la Prueba de Turing

Alan M. Turing propuso el **Test de Turing** en 1950 como medio para proveer una definición operacional satisfactoria de inteligencia. Turing define el comportamiento inteligente (observable) como la habilidad para lograr un desempeño a nivel del humano en todas las tareas cognitivas de tal manera que se pudiera engañar a un interrogador. La prueba consiste en interactuar un agente humano con el sistema artificial (sin que sepa el humano a priori con quién está interactuando) a manera de una sesión de *chat*. El sistema pasa la prueba si el humano no puede discernir si el interactuante es artificial o no. La pruba ha germinado toda una gama de áreas de investigación y desarrollo pragmático, ya que el sistema de cómputo debe poseer las siguientes capacidades:

1. Procesamiento de lenguaje natural,
2. Representación del Conocimiento,
3. Razonamiento mecánico,

4. Aprendizaje automático.

Adicionalmente, en la Prueba completa de Turing (la cual toma en consideración la interaccuón física directa con el interrogador) involucra las áreas de Robótica (manipulación, movimiento) y Visión computacional.

El aspecto de actuar como humano tiene vital importancia en sistemas que necesitan interactuar con los humanos: tutores, sistemas expertos para diagnóstico médico (explicar cómo infirió un diagnóstico), sistemas para procesamiento de información en lenguaje natural, etc.

Pensamiento humano: enfoque cognitivo

Ésta área tiene como propósito *introducirse* dentro del funcionamiento real de la mente humana. Se consideran dos formas para ello:

- *Intronspección* intentar atrapar los pensamientos tal y como ocurren,
- *Experimentos psicológicos*

Una vez teniendo una teoría *precisa* de cómo funciona la mente, se puede construír un programa de computadora que exprese esa teoría: si el comportamiento del programa ante las entradas y salidas empata el comportamiento humano acertadamente (en el transcurso del tiempo) entonces se tiene evidencia que algunos de los mecanismos del programa también operan en los humanos. El campo interdisciplinario de la *ciencia cognitiva* busca conjuntar los modelos computacionales de IA y las técnicas experimentales de la psicología para intentar construír teorías estables y precisas del funcionamiento de la mente humana.

Pensamiento Racional: leyes de razonamiento

Aristóteles fué uno de los primeros filósofos en intentar codificar el pensamiento como procesos de razonamiento irrefutable; sus *silogismos* describen patrones para estructuras de argumentos que siempre dan como resultado conclusiones correctas dadas como entrada premisas correctas. Se supuso que tales leyes generalizadas de pensamiento gobernaban la operación de la mente, iniciando la *lógica simbólica.* Intuitivamente, el propósito fundamental de la lógica es ofrecer una notación precisa para describir enunciados que se refieren a su vez de todo tipo de cosas en el mndo, así como capturar las relaciones que ocurren entre ellas. Alrededor de 1965 se construyeron programas que dado suficiente tiempo y memoria, y dados como entrada la descripción de un problema en notación de lógica encontraban la solución al problema siempre que existiese alguna (si no hay solución, el programa podría en principio continuar buscando por siempre). A ésta línea de investigación se le conoce como *formal clásica* (tradicional), y se basa en la construcción de ése tipo de programas para crear sistemas inteligentes.

Uno de los principales obstáculos en ésta tradición es que en general no es fácil formalizar el conocimiento informal, y más aún cuando el conocimiento es incierto. Además, hay una diferencia enorme a menudo entre encontrar una solución en principio a un problema y resolverlo propiamente en la práctica. Aún cuando ambos obstáculos se presentan ante cualquier intento de construír sistemas de razonamiento computacional, se presentaron primero en la tradición simbólica formal debido al poder de la representación y el razonamiento están bastante bien definidos y entendidos.

Acción Racional: el enfoque de Agente Racional

Esencialmente, actuar racionalmente significa lograr las metas individuales dadas las propias creencias. Bajo éste enfoque, el propósito de la IA es el estudio y construcción de Agentes racionales. Un agente es en esencia una entidad situada que percibe y actúa (en la siguiente sección discutiremos los tipos de agentes).

Nótese que el lograr inferencias correctas (enfoque de pensamiento racional) es tan sólo un aspecto del ser agente racional, dado que una forma para actual racionalmente es razonar lógicamente hacia la conclusión de que una acción determinada permitirá alcanzar alguna meta y entonces actuar sobre tal conclusión. Sin embargo, la inferencia correcta no necesariamente es todo el aspecto de racinalidad debido que existen situaciones en las cuales no existe una acción *demostrablemente correcta*, aún cuando deba hacerse algo de cualquier forma: quitar la mano de algo caliente es una acción por reflejo y en general es más exitoso que deliberar cuidadosamente.

Todas las habilidades cognitivas presentes en la Prueba de Turing permiten acciones racionales, y todas ellas ofrecen un aspecto en común: enriquecer el desempeño del agente gracias a la **interacción** con su medio.

Las ventajas que podemos apreciar del enfoque Racional son, entre otras:

- El concepto de racionalidad que se ha esbozado es más general que el enfoque basado en sólo un conjunto de "reglas de razonamiento", ya que aún cuando la inferencia correcta es un mecanismo extremadamente útil, puede ser no absolutamente necesario o exclusivo del comportamiento inteligente.

- Éste enfoque es más ameno para desarrollo científico que los enfoques basados estrictamente en el comportamiento ó el pensamiento humano, debido a que el *estándar* de racionalidad se define claramente y es completamente general. En contraste, el comportamiento humano está bien adaptado para un ambiente específico y es en parte el producto de un proceso evolutivo complejo (en muchos aspectos no completamente conocido) y que seguramente dista mucho de la perfección.

Por tanto, en éste curso discutiremos los principios generales de los agentes inteligentes y las técnicas para construírlos (como programas de software).

1.1 Agentes Inteligentes

El concepto básico de *Agente* que utilizaremos es el siguiente: es un sistema computacional (software -hardware)*situado* en algún ambiente, capaz de realizar *acción flexible autónoma* con el propósito de alcanzar sus *objetivos de diseño (metas).*

Dentro de esta definición existen 3 propiedades claves que se detallan:

- **Autonomía:** el agente tiene la capacidad de actuar sin la intervención directa de otros agentes, así como tener control de sus propias acciones y su estado interno, tal control depende del conocimiento adquirido en base a la experiencia (involucrando cierta capacidad de aprendizaje). El conocimiento que el agente posea de manera interconstruída (inherente, *bult-in*) condiciona su autonomía. Un propósito de diseño de Agentes que se discutirá más adelante es *cómo mantener un balance entre el conocimiento que se programa en el agente y su habilidad para aprender nuevo conocimiento.*

- **Flexibilidad:** esta capacidad de comportamiento involucra a su vez los siguientes conceptos:

– *Reactivo:* el agente es sensible y **reacciona** oportunamente a los estímulos que recibe y cambios que ocurren en su entorno.
– *Pro-activo:* el agente exhibe comportamiento oportunístico conducido por **objetivos** y toma la iniciativa cuando es apropiado. **Planifica** sus actividades para alcanzar sus metass tanto a corto como largo plazo.
– *Social:* el agente tiene la capacidad de **interactuar** cuando sea apropiado con otros agentes (humanos inclusive) para completar sus problemas y ayudar a resolver a los demás con sus actividades propias, i.e. puede modificar su comportamiento en respuesta a los demás. La interacción se fundamenta en la *comunicación* (lenguajes y protocolos).

Estos elementos tienen como consecuencia en el desarrollo de la *adaptabilidad* del agente con su entorno (involucrando otros aspectos como *mobilidad*, por ejemplo).

- **Localización:** el agente tiene un cuerpo y está situado en un ambiente del cual percibe mediante sus sensores y puede realizar acciones que afecten al entorno en alguna forma.

Podemos por tanto sintetizar nuestra definición general de agente de la siguiente forma: un **agente** es un ente **corpóreo** (*físico*, ya sea en software-hardware) que se encuentra **situado** en un **ambiente**, al cual *percibe* a través de **sensores** y *actúa* en consecuencia afectando al entorno mediante sus **efectores**. Su **comportamiento**, expresado por el conjunto de interacciones con el entorno, está encaminado a satisfacer sus **propósitos**[2].

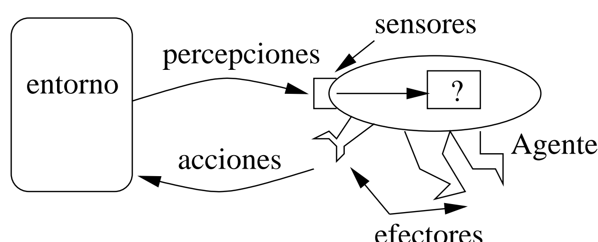


Figura 1.3: Los agentes interactúan con su entorno mediante efectores y sensores para satisfacer sus propósitos.

Por consiguiente, un **Sistema Multiagente (MAS)** está constituído por un conjunto heterogéneo de agentes interactuantes, permitiendo la representación de problemas que involucran múltiples métodos de solución, perspectivas o entidades para solución. De manera intrínseca, tales sistemas tienen las ventajas tradicionales en la solución de problemas concurrente y distribuido, con la ventaja adicional de incorporar *patrones de interacción sofisticados*:

- *Cooperación:* trabajar juntos hacia un propósito común.
- *Coordinación:* organizar las actividades para solución de problemas de tal forma que las interacciones 'daniñas' (en conflicto mutuo) sean evitadas y las interacciones benéficas sean explotadas.
- *Negociación:* llegar a acuerdos aceptables por todas las partes involucradas.

[2]Nuestras definiciones sintetizan adecuadamente aquellas de [NRJ98, RN95, ST99]

La flexibilidad y naturaleza de alto nivel de abstracción de estas interacciones distinguen a los sistemas MAS.

Las características operacionales descriptivas de los MAS son [?]:

- Cada agente tiene capacidades o información *incompleta* para resolver el problema: su punto de vista es limitado.
- No existe un control global.
- La información (datos) y el conocimiento están distribuidos.
- El cómputo es *asíncrono*.

Podemos resumir los requerimientos fundamentales para permitir planificación efectiva en MAS de la siguiente forma:

- Heterogenidad: el entorno MAS debe incluír distintos tipos de agentes. Podemos considerar si es posible a un nivel básico el poder implantar un tipo de agente abstracto ('genérico') a partir del cual construír una amplia gama de agentes especializados[3].
- Cooperación: el entorno *debe* facilitar la cooperación entre los agentes, lo cual yace en principio en los mecanismos de coordinación (operacionalmente: comunicación, exclusión mutua, etc.) y por consiguiente en *métodos de interacción* eficiente entre los agentes.
- Coherencia: asegurar actividades coherentes y validar el comportamiento global del sistema; para ello, es el uso de un modelo formal (riguroso) para especificar las actividades de los agentes (en diversos niveles de abstracción -como individuos independientes y como sociedades estructuradas-), así como técnicas formales para verificar las interacciones entre (e *intra-acciones*) de los agentes.

Las *metas* que persiguen los agentes son especificadas mediante una medida de desempeño[4] la cual mide su grado de éxito. Los agentes pueden clasificarse por los tipos de tareas que realizan, el ambiente en el cual se desempeñan y las metas que los conducen.

Un **Agente racional** es aquel que hace siempre lo correcto. Lo correcto, o la acción correcta, es aquella que hace que el agente sea más exitoso. Existen varios problemas con esta definición:

- ¿Cómo medir el éxito? (contar con una medida de desempeño externa y objetiva)
- ¿Cuándo medir el éxito? (a corto o largo plazo)

En algunos casos se confunde la racionalidad con la *omnisciencia* la cual da la capacidad de conocer el efecto de todas las acciones Obviamente la omnisciencia garantiza el éxito del agente, pero la racionalidad no depende del éxito sino del empleo de mecanismos racionales de toma de decisiones.

Ilustremos ésta idea mediante el ejemplo propuesto por Russell y Norvig:
Un agente va caminando por la calle y ve en la otra acera a un amigo a quien desea saludar, entonces decide cruzar la calle y antes de hacerlo mira a ambos lados de la calle para

[3]En el mismo espíritu que ofrece la programación orientada a objetos, en su modalidad *basado en prototipos*.

[4]Se consideran como *racionales* si los agentes actúan para maximizar la probabilidad de lograr sus metas [?].

	P	**A**	**G**	**E**
Sistema de diagnóstico médico	Síntomas, evidencias y respuestas del paciente	Preguntas al paciente, exámenes y tratamientos	Paciente sano, mínimo costo	Paciente, hospital
Tutor inteligente	Palabras ingresadas por teclado	Ejercicios sugeridos, corecciones	máximo puntaje	Estudiantes
Robot movil	sensores de contacto, mapa de pixels	girar, avanzar, retroceder	Ruta	Oficina

Tabla 1.2: Algunos ejemplos de descriptores PAGE

asegurarse que no viene ningún coche. Cuando va por la mitad de la calle, un piano que cayó del piso 10 de un edificio lo aplasta e impide llegar a la otra acera...

La acción de cruzar la calle fué racional, independientemente del éxito; de hecho otro agente, equipado con radar, que haya esquivado al piano **no** es más racional.

Por lo antes expuesto, debemos afinar nuestro concepto de racionalidad y empezaremos por decir que depende de varios factores:

- La medida de desempeño.
- La secuencia de percepciones del agente.
- Lo que el agente sabe acerca del ambiente.
- Las acciones que el agente puede realizar.

Un **Agente racional ideal** es aquel que para cualquier secuencia de percepciones realiza la acción que maximiza su medida de desempeño sobre la base de la evidencia provista por la secuencia de percepciones y el conocimiento que posee acerca del ambiente.

La implantación de la descripción del agente es un *programa* que debe poder ejecutarse en alguna **arquitectura**.

Agente = un programa + una arquitectura

Russell y Norvig proponen un esquema taxonómico para los agentes, basado en 4 descriptores: **Percepciones, Acciones, Metas y Ambiente** (PAGE).

Los descriptores P y A son internos al agente y forman parte de su arquitectura, mientras que los descriptores G y E son externos. Con respecto a las metas, las cuales codifican la medida de desempeño, deben ser externas ya que de no ser así correríamos el riesgo de adaptar la metas a nuestro comportamiento. El ambiente, como se desprende de los ejemplos, puede ser real o artificial y a diferencia de la opinión de Brooks en [**?**], los ambientes artificiales pueden ser tan o mas complejos que los reales.

El esqueleto de un programa de agente:

```
action function Esqueleto(percepcion)
{       static memoria;
        memoria = ActualizaMemoria(memoria, percepcion);
        accion = Seleccionar(memoria);
```

```
        memoria = ActualizaMemoria(memoria, accion);
        return (accion);
}
```

Existen varias formas de implantar la selección de la acción, pero podemos dividirlas entre las que usan métodos de razonamiento y las que hacen búsqueda sobre tablas de asociación de percepciones y acciones.

Un agente que haga uso de una tabla de asociación podría especificarse así:

```
action function AgenteTabla(percepcion)
{       set perceptos;
        matriz tabla;
        append(percepcion, perceptos);
        accion = buscar(perceptos, tabla);
        return (accion);
}
```

Sin embargo, a utilización de estas tablas trae consigo algunos inconvenientes:

- Las tablas para tareas medianamente complejas suelen ser muy grandes.
- La construcción y depuración de tablas de asociación puede tomar mucho tiempo.
- Limita o anula la autonomía del agente.
- De incorporarse aprendizaje, este sería muy largo (lento).

Para ambientes muy simples este enfoque puede funcionar, pero la pregunta es : ¿Por qué es mejor razonar?

Discutiremos los siguientes enfoques para implantación de los programas de agentes:

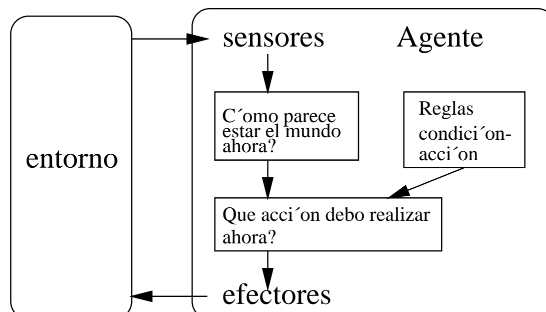
- Agentes reflex (por reflejo)
- Agentes con memoria
- Agentes basados en metas
- Agentes basados en utilidad

1.1.1 Agentes Reflex

Basan la determinación de su estado interno y por ende de su comportamiento exclusivamente en la percepción más reciente, no mantienen historia de las percepciones.

La descripción (funcional) de su comportamiento es:

```
action function AgenteReflex(percepcion)
{       set reglas;
        estado = interpretar(percepcion);
        regla = match(estado, reglas);
        accion = reglas[regla].accion;
        return (accion);
}
```

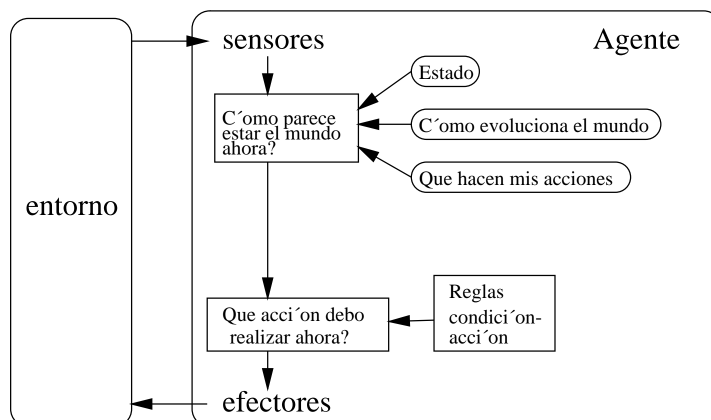


Para la determinación de la acción se utiliza la regla que resulta de la aplicación de la función `match`, la cual además de seleccionar las reglas aplicables realiza la resolución de conflictos.

1.1.2 Agentes con memoria

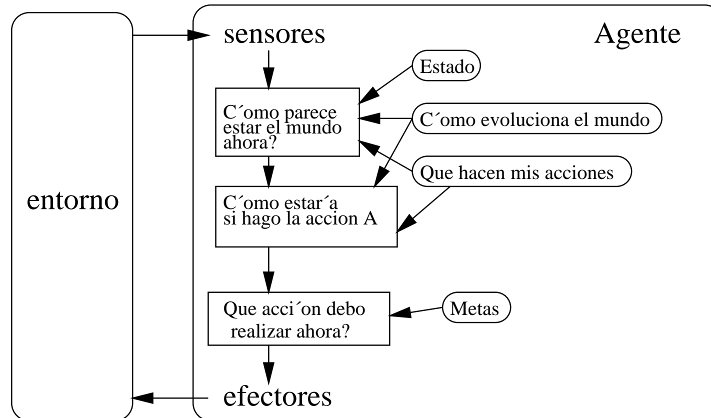
Mantienen un estado interno basado en la historia de percepciones y un modelo del ambiente; este modelo suple en parte las deficiencias perceptivas en términos de discenir cual es el verdadero estado del ambiente. Igualmente este modelo le da al agente una idea de cómo evoluciona el ambiente y cual es el efecto de sus acciones.

```
action function AgenteMemoria(percepcion)
{       set reglas;
        set estado;
        estado = ActualizarEstado(estado, percepcion);
        regla = match(estado, reglas);
        accion = reglas[regla].accion;
        estado = ActualizarEstado(estado, accion);
        return (accion);
}
```



1.1.3 Agentes basados en metas

Establecer metas condiciona la selección de acciones. En ocasiones una meta se alcanza por la ejecución de una sola regla; pero en la mayoria de los casos debe ejecutarse una **secuencia de acciones** lo que requiere de capacidad de búsqueda y planificación. Esto trae consigo mucha más flexibilidad, a cambio de una disminución en la eficiencia. Un agente basado en metas debe aplicar sus reglas a conjuntos de estados y construír un plan basado en el logro de la meta planteada.



La función general del agente es muy similar a la del agente con memoria y la diferencia está en los criterios para selección de la acción y la utilización del modelo de ambiente y del modelo de acción como elementos de planificación.

1.1.4 Agentes basados en utilidad

En este caso los agentes no sólo deben lograr una meta, sino que tienen que incrementar su medida de desempeño, para lo cual hacen uso de una función de utilidad de la forma:

$$estados \xrightarrow{f} \mathcal{R}$$

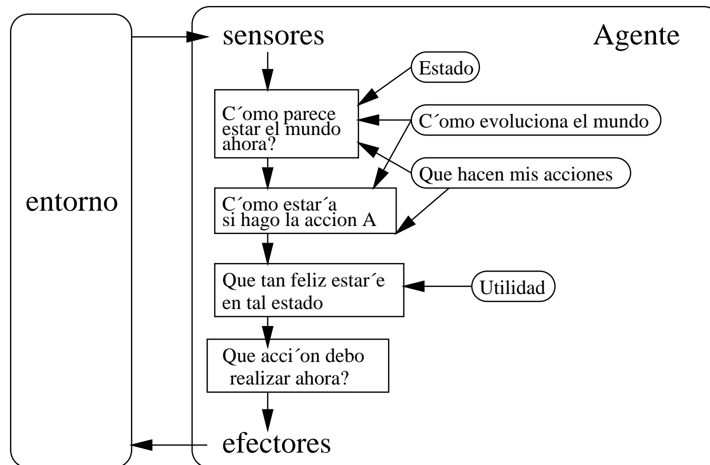
Dicha función de utilidad asocia los estados a valores reales que representan la utilidad de dichos estados, o dicho de otra forma, el grado de bondad del estado para el agente. Más que representar un problema, la función de utilidad sirve para solventar problemas de conflictos entre múltiples metas o restricciones.

Para efectos de implantación puede convertirse la función de utilidad en un conjunto de metas y utilizar un agente basado en metas para resolver el problema. Otra posibilidad es la de evaluar la utilidad de los pares acción/estado, basados en la utilidad del estado al cual trasladan propiamente al agente.

1.1.5 Ambientes

Los ambientes pueden clasificarse de acuerdo a varios criterios:

- **Accedible**: los sensores dan el estado completo del ambiente ó al menos todos los estados relevantes para la toma de decisiones. En un ambiente accedible no es necesario el mantenimiento de estados internos.



- **Determinístico**: el estado futuro del ambiente depende completamente del estado actual y de la acción tomada. Un ambiente accedible y determinístico nos libera de incertidumbre. El determinismo siempre debe evaluarse desde el punto de vista del agente ya que el problema de accedibilidad del ambiente pueden hacer parecer no determinísticos a algunos ambientes que en realidad son determinísticos.
- **Episódico**: existen fases de percepción y acción independientes. El agente no nececesita planificar más allá del episodio actual.
- **Estático**: el ambiente no cambia durante la deliberación del agente. En el caso en que el ambiente no cambia pero la medida de desempeño si, decimos que el ambiente es semi-dinámico.
- **Discreto**: el número de percepciones y acciones es limitado.

El peor caso lo constituyen los ambientes inaccesibles, no episódicos, dinámicos y contínuos. Con respecto al determinismo, en la práctica, la complejidad de los ambientes interesantes hacen preferible su tratamiento no determinístico. Podemos esbozar dos funciones generales del ambiente, las cuales rigen su dinámica; la segunda de ellas realiza además una evaluación del desempeño de los agentes en el ambiente:

```
void function Ambiente(estado, actf, agentes, termina)
{
    for(;!termina(estado);); {
    for (agente=0; agente<agentes.length; agente++) {
        percep[agente] = sensar(agentes[agente], estado);
        accion[agente] = agentes[agente].programa(percep[agente]);
        }
    estado = actf(accion, agentes, estado);
    }
}


array function AmbienteEval(estado, actf, agentes, terminacion, desf)
```

```
{array scores; /* arreglo local */
    for(;!terminacion(estado);); {
    for (agente=0; agente<agentes.length; agente++) {
        percep[agente] = sensar(agentes[agente], estado);
        accion[agente] = agentes[agente].programa(percep[agente]);
        }
    estado = actf(accion, agentes, estado);
    scores = desf(scores, agentes, estado);
    }
}
```

Es conveniente recordar que el estado del ambiente **es diferente** al estado del agente.

Tarea 1

1. Describe en términos de los descriptores PAGE los siguientes sistemas Agente:
   (a) Virus de computadora
   (b) Monitor de tráfico en el crucero del Blvd. Valsequillo y 14 sur.
   (c) Sistema Tutor que permita a los alumnos participar en foros *en-línea.*

   Indica y justifica claramente el ambiente en el cual debe desenvolverse el/los agentes, sus metas, etc. de tal manera que no quede duda alguna para su implantación computacional (que sería una tarea futura).

2. Para las descripciones anteriores, disute cuál es el tipo de arquitectura que más conviene: reactivo, deliberativo, etc.

   Fecha límite de entrega: **Martes, Sept. 10**.

3. Lee los capítulos 1, 26 y 27 del libro de texto:

   Artificial Intelligence, a modern approach
   Stuart Russell and Peter Norvig
   Prentice Hall

   Prepara un resúmen (personal) con vías al primer parcial: **Sept. 26**.

Capítulo 2

Técnicas de Búsqueda y heurísticas

En éste capítulo discutiremos el diseño de agentes basados en metas que deciden sistemáticamente qué hacer, considerando los resultados de las distintas secuencias de acciones disponibles para resolver un problema. Primero plantearemos los elementos a considerar: planteamiento del problema, tipos de problemas, planteamiento de la meta (objetivo a alcanzar); y discutiremos una serie de técnicas para resolver problemas aplicando **búsqueda** sobre un *grafo* el cual representa los *estados del problema.* Los tipos de problemas que resultan del proceso de planteamiento dependen del conocimiento disponible al agente: si conoce el *estado actual* así como los *resultados de sus acciones.*

Primero discutiremos un ejemplo intuitivo para familiarizarnos con el concepto y entonces analizaremos las definiciones formales.

Ejemplo 1 *En la orilla de un río se encuentra un granjero, junto con un lobo, una oveja y paja. Hay un bote con la capacidad suficiente para llevar al hombre y a uno de los otros tres. El hombre con la paja, y demas compañeros deben cruzar el río, y el hombre puede llevar a uno solo a la vez. Sin embargo, si el hombre deja solos al lobo y a la oveja en cualquier lado del río, con toda seguridad que el lobo se comerá a la oveja. Del mismo modo, si la oveja y la paja se quedan juntas, la oveja se comerá a la paja. ¿Es posible que se pueda cruzar el río sin que se pierda alguno?*

Podemos modelar el problema apartir de la observación que la información necesaria es en **dónde** se encuentran los ocupantes después de que se efectúa la transición (acción) de cruzar el río: el problema se habrá resuelto cuando se haya elegido una secuencia de transiciones que permita tener a todos los pasajeros en el extremo contrario del río del cual se comenzó. Por tanto, describiremos los estados mediante parejas de conjuntos separadas por un guión, cada conjunto representa a quienes están en ésa orilla del río: Hombre (H), Oveja (O), Paja (P), Lobo (L). Así, el **estado inicial** del problema es la tupla

$$\langle H, L, O, P - \emptyset \rangle$$

Nótese que en la parte derecha de la tupla se tiene el conjunto vacío para indicar que ninguno de los pasajeros está en ese extremo del río.

Algunos estados son fatales y deben evitarse, por ejemplo

$$\langle L, O - H, P \rangle$$

ya que el lobo se come a la oveja.

Como mencionamos, las acciones que el hombre toma para atravesar el río son las **transiciones de estados**: puede cruzar solo en la barca (H) ó elegir como pasajero al lobo (L), a la oveja (O) ó a la paja (P). Por ejemplo

$$\langle H, L, O, P - \emptyset \rangle \xrightarrow{L} \langle O, P - H, L \rangle$$

indica que el hombre tomó al lobo como acompañante para cruzar el río. Podemos nombrar a cada transición indicando el pasajero adicional exclusivamente[1].

La meta, propiamente el **estado final** ó **estado de aceptación** en el cual el problema está resuelto es justamente el *dual* al estado inicial, i.e. todos los pasajeros en el otro extremo del río:

$$\langle \emptyset - H, L, O, P \rangle$$

Para resolver el problema se tiene que elegir una secuencia de movimientos (atravesar el río) que a partir del estado inicial se alcance el estado final. Esa secuencia se puede representar *concatenando* las iniciales de cada pasajero que atravesó el río, i.e. las acciones (transiciones) que se eligen.

Una posible secuencia (solución al problema) es la siguiente :

$$\begin{array}{c}
\langle H, L, O, P - \emptyset \rangle \xrightarrow{O} \langle L, P - H, O \rangle \\
\langle L, P - H, O \rangle \xrightarrow{H} \langle H, L, P - O \rangle \\
\langle H, L, P - O \rangle \xrightarrow{P} \langle L - O, H, P \rangle \\
\langle L - O, H, P \rangle \xrightarrow{O} \langle H, L, O - P \rangle \\
\langle H, L, O - P \rangle \xrightarrow{L} \langle O - H, L, P \rangle \\
\langle O - H, L, P \rangle \xrightarrow{H} \langle H, O - L, P \rangle \\
\langle H, O - L, P \rangle \xrightarrow{O} \langle \emptyset - H, L, P \rangle
\end{array}$$

Lo cual podemos indicar de la siguiente manera:

$$\langle H, L, O, P - \emptyset \rangle \xrightarrow{*} \langle \emptyset - H, L, O, P \rangle$$

Podemos representar el problema mediante un **grafo de transiciones**, como se indica en la Figura 2.1, en la cual se puede observar todas las posibles elecciones que estando en algún estado se puedan tomar. El estado final se encuentra resaltado. Puede apreciarse que existen dos soluciones que son las más cortas (i.e., tienen el mismo tamaño de la ruta correspondiente) pero en general hay un número infinito de soluciones al problema las cuales involucran pasar las veces que se quiera por los mismos estados (i.e *ciclos inútiles*).

Recordemos que un agente racional guía su comportamiento a partir de la suposición de que puede provocar efectos en el ambiente de tal suerte que pase a través de una secuencia de estados que le permitan al agente maximizar su medida de desempeño. La tarea se simplifica si el agente adopta ó plantea **metas** que al satisfacerlas tenga como resultado tal incremento en el desempeño. La formulación ó planteamiento de metas *basadas en la situación actual* es el primer paso para resolver un problema: que se quiere/necesita hacer. A continuación, el agente debe *decidir* qué otros factores que afectan el atractivo ó conveniencia de las formas distintas de alcanzar la meta.

Consideraremos, sin pérdida de generalidad, que una meta se describe mediante *un conjunto de estados del ambiente* y en cualquier estado dado el agente puede *evaluar* si cumple

[1]Obviamente, ni el lobo ni la oveja y mucho menos la paja pueden controlar la barca...

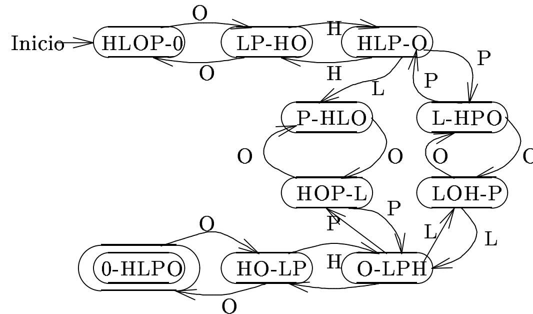


Figura 2.1: Diagrama de transiciones para el problema de cruzar el río.

o no la condición de meta. Las acciones causan transiciones entre los estados del ambiente, por tanto, el agente debe identificar qué secuencia de acciones conducen a la meta. Las acciones deben establecerse en un tipo de detalle útil: por ejemplo, para cruzar el río no tiene sentido mencionar transiciones como "mover un brazo 30 cm. hacia la oveja ", "mover un pie", etc.

El proceso de **planteamiento del problema** consiste justamente en decidir qué tipo de acciones y estados del ambiente se van a considerar y es un proceso inmediato tras plantear la meta.

Se denomina **búsqueda** al proceso de identificar entre las posibles secuencias de acciones (transiciones de estados) la mejor que conduce a estados meta. Un algoritmo de búsqueda recibe como entrada el planteamiento de un problema y regresa como salida la **solución** en términos de una secuencia de acciones. Se puede identificar, por tanto, el ciclo clásico:

percibe $\longrightarrow$ **formula-problema** $\longrightarrow$ **busca-solución** $\longrightarrow$ **ejecuta** $\longrightarrow$ **percibe ...**

Tal es el comportamiento del agente cuyo esqueleto se muestra a continuación:

```
action function agenteSimpleSolProb(percepcion)
{       set seqacciones;
        set estado;
        goal meta;
        problem problema;
        estado = ActualizarEstado(estado, percepcion);
        if vacio(seqacciones) {
                meta = FormulaMeta(estado);
                problema = PlanteaProblema(estado, meta);
                seqacciones = Buscar(problema);
        }
        accion = Primero(seqacciones, estado);
        seqacciones = Resto(seqacciones, estado);
        return (accion);
}
```

Nótese sin embargo que el agente *planifica* (i.e., el proceso de buscar una secuencia de acciones) únicamente cuando no hay una acción a realizar en la lista `seqacciones`; si el ambiente cambia drásticamente podría no ser efectiva la acción siguiente en la lista y en cuyo caso se deban reformular las acciones a realizar.

2.1 Planteamiento del Problema

En general, podemos considerar los siguientes tipos de problemas:

Uni-estado cuando el ambiente es completamente accedible, el agente sabe con precisión en qué estado se encuentra y sabe exactamente qué hacen sus acciones.

Multiestado cuando el entorno no es completamente accedible al agente pero sabe qué efectos causan sus acciones: el agente debe razonar y guiar su comportamiento considerando conjuntos de estados en lugar de un único estado a la vez[2].

De Contingencia cuando resulta imposible predecir con exactitud en qué estado estará el ambiente.

Exploración cuando el agente no inicia con conocimiento sobre el efecto de sus acciones y debe experimentar gradualmente para descubrir lo que causan sus acciones y qué clases de situaciones existen.

Definimos a un **Problema** como una colección de información que el agente usa para decidir qué hacer. Centraremos nuestra atención hacia problemas uni-estado:

Estado Inicial: aquel en el cual el agente sabe que se encuentra.

Operadores: las acciones disponibles del agente se describen indicando de qué estado parte y qué estado será alcanzado al aplicar la acción. También se denomina función $Sucesor(s)$ donde s es el estado actual.

Espacio de búsqueda: ó también, espacio de estados[3], está dada por los elementos anteriores: todos los estados *alcanzables* partiendo del estado inicial y aplicando alguna secuencia de acciones.

Condición de meta: el agente aplica una prueba a la descripción de estado para determinar si cimple o no la propiedad ser un estado meta.

Función de costo: a menudo una solución es preferible a otra (e.g., la más corta) o las acciones que involucra sean más costosas que otras; una función de costo asigna un costo a una ruta, la cual en general resulta de la suma de los costos de las acciones individuales a lo largo de la ruta. Denotaremos a la función de costo por g.

Solución: es una ruta desde el estado inicial hacia algún estado que satisface la condición de meta.

Naturalmente, se puede construír una estructura de datos para representar problemas, así como la utilización de grafos para representar el espacio de estados: los nodos denotan estados y los operadores denotan aristas (arcos) entre estados. El proceso de búsqueda por una solución se reduce a encontrar una ruta entre el nodo inicial hacia el nodo meta del grafo.

La *efectividad* del proceso de búsqueda se puede medir en tres aspectos:

1. ¿Encuentra una solución?
2. ¿Encuentra una buena solución (a bajo costo)?

[2]Una analogía útil es considerar que los problemas uni-estado se les puede caracterizar mediante autómatas finitos determinísticos, mientras que los multiestado como autómatas finitos no determinísticos.

[3]Para problemas multiestado, corresponde a espacio de conjuntos de estados.

3. ¿Cual es el costo de búsqueda (asociada con el tiempo y almacenamiento en memoria)? Se considera que el costo *total* de la búsqueda estará dado por la suma del costo de la ruta (*online*) y el costo de búsqueda (*offline*).

El verdadero arte en la solución de problemas es **abstraer** lo que resulta **necesario y útil**, omitiendo los detalles prescindibles, lo cual se aplica tanto en las descripciones de estados como en las acciones mismas.

Podemos identificar dos clases generales de problemas a discutir: los problemas *juguete* que tienen el propósito de ilustrar los métodos de solución de problemas, se pueden caracterizar de manera conscisa, exacta y a menudo se utilizan como referencia para comparar el desempeño de los algoritmos de búsqueda. Por su parte, los problemas *del mundo real* son sumamente complejos, no tienen una descripción totalmente acordada y su solución tiene impacto (a nivel industrial, económico, social, etc.).

Como ejemplo de problemas de juguete, tenemos el rompecabezas que consiste de un tablero de 3×3 espacios con 8 fichas. Las fichas adyacentes al espacio se pueden desplazar ocupando el espacio. Un planteamiento del problema es:

Estados: localización de las fichas en los 9 cuadros del tablero, así como la localización del espacio vacío.

Operadores: desplazar el espacio a la izquierda, derecha, arriba ó abajo.

Condición de meta: el estado coincide con la configuración que se muestra a la derecha de la Figura 2.2.

Costo de la ruta: cada paso cuesta 1, el costo de la ruta es simplemente la longitud de la misma.

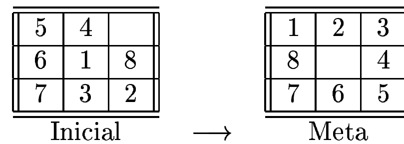


Figura 2.2: Rompecabezas de 8 fichas.

Por su parte, entre los problemas del mundo real más representativos están el encontrar caminos: navegación robótica, ruteo de paquetes en redes de computadoras, sistemas de agencia de viajes (planificación de vuelos), etc. El problema del **Agente viajero** consiste en visitar, al menor costo, todas las ciudades de un conjunto partiendo y regresando de una ciudad dada. Sólo se permite pasar por una ciudad una sola vez (a excepción de la ciudad inicial que es la destino final). Éste problema es referencia obligatoria para probar nuevos algoritmos.

Ejercicios 1

1. *Describe los componentes del problema de cruzar el río.*
2. *Para cada uno de los descriptores PAGE de la tarea 1, primer inciso, identifica un problema y descríbe sus componentes.*

2.2 Estrategias de Búsqueda

La idea esencial de un proceso de búsqueda es mantener y extender un conjunto de secuencias (soluciones parciales) de descripciones de estados. Dado el nodo (estado) actual, se verifica la condición de meta; si no la cumple entonces se elige una acción, el aplicar el operador correspondiente **genera** ó conduce a nuevos estados; éste proceso se denomina **expansión de estados**. Se continúa eligiendo, probando y expandiendo hasta encontrar una solución ó ya no existan más estados que se puedan expander. La elección de cuál estado deba expanderse a continuación la define la **estrategia de búsqueda**.

Es útil considerar el proceso de búsqueda como la construcción de un *árbol de expansión*: la raíz es el nodo inicial. A cada paso, el algoritmo elige un nodo hijo a expander, cuyos sucesores serán a su vez visitados. Las hojas son nodos que ya no tienen sucesores (no se pueden expander ó ya han sido visitados).

```
solution function BusquedaGral(problema, estrategia)
{       /* regresa fallo si no hay solucion */
        set psol;
        list bLista, hijos;
        node nodo;

        bLista = inicial(problema);
        while not vacia(bLista) {
                do {
                        nodo = primero(bLista);
                        bLista = resto(bLista);
                }while  visitado(nodo)
                if meta(nodo) return psol;
                incorpora(nodo, psol);
                expander(nodo, hijos);
                inserta(hijos, bLista, estrategia);
        }
        return (fallo);
}
```

Asumiremos que la estructura de datos para representar un nodo contiene la siguiente información:

1. el estado del espacio de búsqueda,
2. un enlace a su nodo padre, (que lo generó)
3. el operador que fué aplicado para generar éste nodo,
4. un indicador si ya ha sido visitado o no,
5. la profundidad (núm. nodos ancestros desde la raíz),
6. el costo de la ruta que lleva a éste nodo.

No hay que confundir nodos con estados del mundo. Se supone que la función `expander` actualiza la información correspondiente de cada nodo generado. La estrategia de búsqueda

esencialmente elige el siguiente nodo a ser expandido de la lista de nodos que están esperando a ser expandidos.

Las estrategias de búsqueda se evalúan conforme a los siguientes criterios:

- Completitud: si la estrategia segura encontrar una solución siempre que ésta exista.
- Complejidad en el tiempo: cual es la duración para encontrar una solución.
- Complejidad en el espacio: cual es la demanda de almacenamiento.
- Optimalidad: si la estrategia elige la mejor solución.

A continuación se describen estrategias de búsqueda *sin información* ("a ciegas") ya que no se consideran los pasos ó el costo de una ruta. Las estrategias que si toman en cuenta tales aspectos se denominan con información ó **heurísticas**.

Las estrategias que se presentan a continuación difieren fundamentalmente del *orden* en el cual los nodos son expandidos. Sin embargo, tal diferencia trae consigo repercusiones en los 4 criterios anteriores.

2.2.1 Primero en Profundidad (PP)

Ésta estrategia expande siempre uno de los nodos a la mayor profundidad del árbol (i.e., sigue una ruta hasta encontrar nodos hojas). Illustraremos a continuación esta estrategia a partir de un grafo que nos representa el problema de encontrar un camino entre dos ciudades[4]:

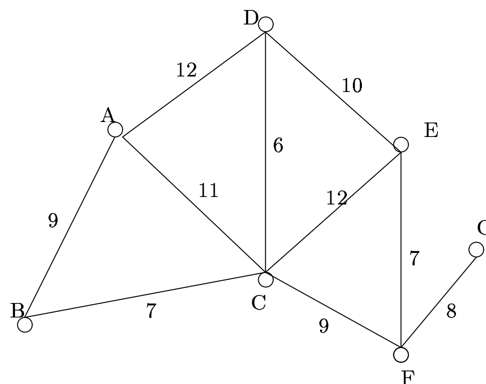


Figura 2.3: Grafo/mapa para encontrar una ruta entre dos ciudades. Los valores en las aristas indican un costo (por ejemplo, el kilometraje).

Consideremos que el estado inicial es A, mientras que el estado Meta: G.

Esencialmente, PP coloca los nodos a ser expandidos en el frente de la lista de expansión `bLista`, de tal suerte que en la siguiente ronda sean seleccionados a expanderse. Ello trae como consecuencia que se siga una ruta hasta alcanzar hojas (máxima profundidad) antes de intentar otra alternativa.

[4]Por el momento, no consideraremos el costo.

```
solution function primeroProfundidad(problema) {
        return BusquedaGral(problema, Encolar_al_frente)
}
```

Para nuestro ejemplo, se muestra en la Figura 2.4 los pasos que sigue para encontrar una solución. En cada nivel del árbol se muestran los nodos que se consideran a ser expandidos, los arcos con líneas de puntos indican nodos que no necesitan expanderse (hojas) debido a que ya han sido previamente visitados; indicados con flechas se muestran los nodos que forman una ruta hacia la meta (solución).

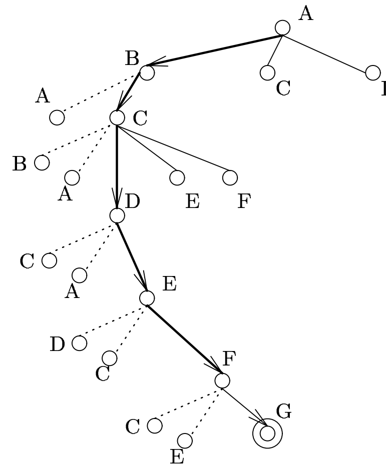


Figura 2.4: Pasos que sigue PP para encontrar una solución.

En cada nivel podemos apreciar cuál es el contenido de la lista de expansión `bLista`: (quedan entre paréntesis aquellos nodos que se descartan -no se encolan- por haber sido visitados previamente):

A
B, C, D
(A), C, C, D
(B), (A), D, E, F, C, D
(A), (C), E, E, F, C, D
(D), (C), F, E, F, C, D
(C), (E), **G**, E, F, C, D

2.2.2 Primero en Amplitud (PA)

Por su parte, PA esencialmente sigue la política de expander todos los nodos candidatos que se encuentran al mismo nivel del árbol de búsqueda antes de explorar otro nivel. Ello se consigue siguiendo la política de encolar al final en la lista de expansión `bLista`:

```
solution function primeroAmplitud(problema) {
        return BusquedaGral(problema, Encolar_al_final)
}
```

Para el problema de la Figura 2.3, los pasos que realizará PA se muestran en la Figura 2.5. Nótese que mientras que PP necesitó explorar el árbol hasta nivel 6, a PA le basta sólo 4 niveles para encontrar una solución.

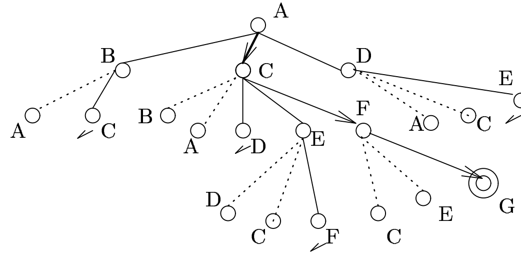


Figura 2.5: Pasos que realiza PA para el problema de encontrar una ruta.

El contenido de la lista de expansión `bLista` se muestra a continuación:

A
B, C, D
(A), C, (B), (A), D, E, F, (A), (C), E
(D), (C), F, (C), (E), **G**

Nótese que algún nodo que haya sido encolado como descendiente de algún otro nodo, puede ser visitado anticipadamente por algún nodo en el nivel anterior, de tal suerte que sea desacartado al descender al siguiente nivel. Por ejemplo, C se encola como descendiente de B en el nivel 3, pero C ya estaba encolado por ser hermano de B en el nivel 2. Cuando todo el nivel 2 ha sido expandido, C ya está marcado como visitado por la tanto ya no se expande otra vez (en la rama originada como sucesor de B).

Mientras que PP obtiene la solución:

$$A \rightarrow B \rightarrow C \rightarrow D \rightarrow E \rightarrow F \rightarrow G$$

PA obtiene una solución (en éste caso más corta):

$$A \rightarrow C \rightarrow F \rightarrow G$$

Mostraremos más comparaciones una vez que presentemos algunas extensiones a las técnicas anteriores.

2.2.3 Algunas extensiones

Costo Uniforme (CU)

PA encuentra el estado meta *menos profundo*, pero ésto no significa que la correspondiente ruta sea la de menor costo. La técnica de búsqueda por costo uniforme es una modificación a PA de tal suerte que siempre selecciona el nodo cuya arista correspondiente es de menor costo. Denotaremos por $g(n)$ la función de costo de una ruta que llega hasta el nodo n.. Entonces, PA se puede ver como búsqueda por costo uniforme con $g(n) = \mathit{profundidad}(n)$.

CU encuentra la solución más económica siempre que se asegure se cumple la siguiente condición:el costo de una ruta nunca debe decrecer conforme se avanza en la ruta: para todo nodo n,

$$g(\mathit{Sucesor}(n)) \geq g(n)$$

Se está considerando que el costo de una ruta es la suma de los costos de los operadores (aristas del grafo) que conectan a la ruta.

Profundidad limitada (PL)

Un tremendo inconveniente de PP es que puede quedarse atorado dentro de una ruta incorrecta y peor aún, dentro de un ciclo potencialmente infinito[5]. Una manera de evitar tales inconvenientes es imponer una cota en la profundidad máxima l a explorar en una serie de rondas. Aun cuando se garantiza encontrar una solución (si tal existe) no se garantiza que se encuentre una solución óptima.

Profundidad iterativa (PI)

El aspecto importante en la búsqueda con Profundidad limitada es elegir una buena cota de la profundidad. Por ejemplo, si tenemos un mapa con 20 ciudades, la primer cota de profundidad que podríamos suponer es 19. Pero si examinamos con cuidado el mapa y vemos que entre cualquier pareja de ciudades existe un máximo de 9 ciudades[6] intermedias, entonces podríamos considerar mejor ese valor de cota. La técnica de PI consiste en elegir la cota de mejor profundidad intentando todas las posibles profundidades, combinando los beneficions de PP y PA:

```
solution function profunIterativa(problema) {
        long profundidad;

        for(profundidad = 0; profundidad < ∞; profundidad++)
                if exitosa(profunLimitada(problema, profundidad))
                return solucion
        return fallo
}
```

PI se prefiere cuando el espacio de búsqueda es grande y la profundidad de la solución no se conoce.

La siguiente tabla muestra la comparación en el desempeño de las estrategias de búsqueda: b indica el factor de ramificación el árbol, m es la profundidad máxima del árbol, l indica la cota de profundidad y d la profundidad ó nivel de la solución en el árbol.

Criterio	PA	CU	PP	PL	PI
tiempo	b^d	b^d	b^m	b^l	b^d
espacio	b^d	b^d	bm	bl	bd
optimalidad	si	si	no	no	si
completitud	si	si	no	si($l \geq d$)	si

Ejercicios 2 *Formula y aplica las estrategias CU, PL y PI para el problema del mapa (Figura 2.3), muestra qué nodos elige en cada paso y cuál es la ruta correspondiente que encuentra.*

[5] Esto ocurre cuando no se marcan los estados (nodos) como previamente visitados, o se visitan estados equivalentes pero el entorno no es completamente acedible para el agente.

[6] A éste valor, si existe, se denomina **diámetro del espacio de estados**.

2.3 Búsqueda con información

En la sección anterior se discutieron técnicas que pueden encontrar soluciones mediante generación sistemática de nuevos estados y verificando si cada uno de ellos cumple la condición de meta. Ahora discutiremos la utilización de información **específica** del problema en particular para encontrar soluciones de manera más eficiente.

2.3.1 Primero el Mejor (PM)

El conocimiento relativo a un problema puede ser útil para decidir qué nodo conviene expandir en cada paso (relativo a estados y operadores), lo cual significa disponer de una **función de evaluación** que mide el atractivo del nodo candidato. Cuando se ordenan los nodos en la lista de expansión de acuerdo a que el nodo con la mejor calificación se expande primero se tiene la estrategia **Primero el Mejor**:

```
solution function primeroMejor(problema, funEval) {
       /* Encolar usa el criterio de funEval */
       return BusquedaGral(problema, Encolar)
}
```

En realidad, PM elige el que *parece* ser el mejor de acuerdo a la función de evaluación. Por lo tanto, ahora se puede considerar toda una familia de algoritmos PM con distintas funciones de evaluación: dado que tienen el propósito de encontrar soluciones de bajo costo, los algoritmos utilizan una *medida estimada* del costo de la solución e intentan minimizarlo.

Aún cuando hemos considerado a la función $g(n)$ como el costo asociado a la ruta, tal función no dirige la búsqueda hacia la meta. Por tanto, la medida **debe incorporar algún estimado del costo de la ruta a partir de un estado hacia el estado meta más cercano**. Mencionaremos dos enfoques:

- Expander el nodo más cercano a la meta,
- Expander el nodo que se encuentre en la ruta de menor costo asociada a la solución.

Búsqueda Ávida

El primer enfoque tiene como propósito general el minimizar el costo estimado para alcanzar la meta, la cual es una de las estrategias PM más sencillas. De esata forma, el nodo que se califique como más cercano a un estado meta se expande primero. Para l amayoría de los problemas, el costo de alcanzar la meta a partir de un estado en particular puede estimarse, pero *no puede determinarse con exactitud*. Una **función heurística** es aquella que permite estimar dicho costo, y se denota por $h(n)$:

$$h(n) = \text{costo estimado de la ruta más barata del nodo } n \text{ hacia un nodo meta}$$

La estrategia PM que utiliza la calificación dada por h como criterio de selección del siguiente nodo a expandir se denomina **Ávida**:

```
solution function bAvida(problema, funEval) {
       /* Encolar usa el criterio de funEval */
       return primeroMejor(problema, h )
}
```

En general, h puede ser *cualquier función*, un requisito importante que debe cumplir es

$$h(n) = 0 \text{ si } n \text{ es meta.}$$

Dado que las funciones heurísticas dependen del problema (rara vez una misma heurística sirve para muchos problemas), ilustraremos la estrategia ávida con el problema planteado en la Figura 2.3 para encontrar una ruta.

Una buena heurística para problemas de encontrar rutas es la **distancia en línea recta** hacia la meta:

$$h_{dlr}(n) = \text{distancia en línea recta desde } n \text{ y la meta}$$

Evidentemente, podríamos ofrecer los valores de h_{dlr} sólo si contamos con un mapa con coordenadas de cada una de las ciudades; en otros casos una estimación (incluso empírica o estadística) sería suficiente:

Ciudad	A	B	C	D	E	F	G
h_{dlr} a G	22	30	14	15	7	7	0

Mostramos en la Figura 2.6 los pasos que realiza ésta estrategia para encontrar una solución. Nótese que la estrategia semrpe prefiere dar el mordisco mas grande posible del costo restante para alcanzar la meta. Lo interesante del problema es que al encontrar dos nodos que coinciden en tener el menor costo (E y F cuando expande C), necesita un criterio adicional para decidir, pues sin mayor informaci'on tomaría primero E, que lo lleva a F y entonces a G, mientras que una solución mejor es elegir primero a F que lleva a G inmediatamente. Esto ilustra el hecho que aún cuando la búsqueda ávida permite encontrar soluciones rápidamente, tales soluciones no son en general las mejores (óptimas): se debe realizar un análisis cuidadoso hacia opciones con repercusiones a largo plazo y no solo la mejor elección inmediata. Otro punto en contra es que la búsqueda ávida es suceptible a *comienzos falsos*: elegir una ruta que conduce a un punto muerto. La búsqueda ávida padece de los mismos defectos

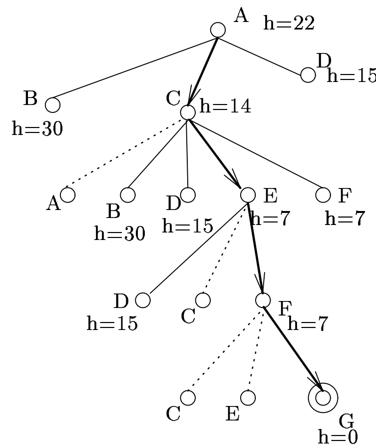


Figura 2.6: Búsqueda Ávida sobre el problema de encontrar una ruta de A hacia G.

que PP: no es óptima ni completa. El peor caso de complejidad en tiempo es $O(b^m)$ para m la profundidad máxima del espacio de búsqueda. Su complejidad en tiempo es igual a su complejidad en espacio debido a que almacena todos los nodos en memoria, con una buena función heurística se puede reducir la complejidad en espacio y tiempo considerablemente.

2.3.2 A^*

La técnica de búsqueda ávida minimiza el costo estimado a la meta dado por $h(n)$; desafortunadamente no es una estrategia óptima o completa. Por su parte, costo uniforme minimiza el costo de la ruta vista hasta el momento $g(n)$, la cual es óptima y completa aún cuando puede ser tremendamente ineficiente. Resulta sensato considerar combinar ambas estrategias, lo cual se puede hacer simplemente sumando[7] las funciones:

$$f(n) = h(n) + g(n)$$
$$f(n) = \text{costo estimado de la solución más barata pasando por } n$$

así se elegirá en cada paso el nodo con el menor valor de f. Se puede demostrar formalmente que esta estrategia es óptima y completa siempre que h *nunca sobre-estime* el costo para alcanzar la meta. Una función h con tal característica se denomina **heurística admisible**, la cual es por naturaleza optimista: *si h es admisible, entonces $f(n)$ nunca sobreestimará el costo real de la mejor solución que pasa por n.* La técnica primero el mejor utilizando f como la función de evaluación dada una función heurística h admisible se conoce como **búsqueda A^***:

```
solution function Aestrella(problema) {
        return primeroMejor(problema, g + h )
}
```

Mostramos en la Figura 2.7 el comportamiento de A^* para el problema de encontrar una ruta. En cada paso, la función f se evalúa de acuerdo al costo (acumulado) de seguir una

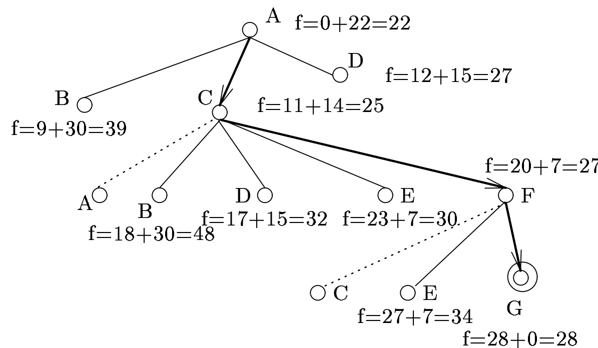


Figura 2.7: Solución encontrada por A^*.

ruta consultando los valores que se encuentran en las aristas de la Figura 2.3 sumando con los valores de la función heurística h_{dlr} de la distancia en línea recta, la cual es claramente admisible debido a que la ruta más corta (euclidiana) entre dos puntos es una recta.

Si en algún paso, en la generación de un nuevo nodo n' vemos que su valor de f es menor que la de su padre n, entonces asumiremos tomar el valor de $f(n)$ (del padre):

$$f(n') = max\big(f(n), g(n') + h(n')\big)$$

esto es con el propósito de mantener *monotonía* y evitar mayores dificultades.

[7] Para preservar monotonía.

La función de costo f permite trazar **contornos** sobre el espacio de búsqueda como si se tratara de un mapa geográfico, compuesto por cuestas y valles. El propósito de A^* es ascender hacia la meta, trazando en el camino la ruta óptima de menor costo. Obviamente la efectividad depende de la función heurística h.

Si definimos como f^* al costo de la ruta solución óptima, entonces ocurre lo siguiente:

- A^* expande todos los nodos n tales que $f(n) < f^*$,
- A^* expanderá entonces algunos de los nodos hacia el contorno meta, para los cuales $f(n) = f^*$ antes de seleccionar un nodo meta.

Dada la importancia de plantear buenas funciones heurísticas para aplicar A^*, conviene discutir brevemente algunas consideraciones útiles mediante un ejemplo clásico: el rompecabezas de 8 piezas mencionado en la Figura 2.2. Ua solución típica para éste problema consta de 20 pasos, aunque ello depende de la **configuración inicial**. El factor de ramificación es alrededor de 3 (i.e., las posibles alternativas en las cuales se pueden mover las fichas: cuando la casilla vacía está en el centro hay cuatro posibles movimientos, cuando está en alguna esquina son sólo dos y cuando está sobre una columna/renglón es tres). Por tanto, una búsqueda exhaustiva con cota de profundidad 20 se enfrenta a un espacio de 3^{20} estados!. Si se eliminan estados repetidos se reduce a $9! = 362,880$ configuraciones, lo cual sigue siendo considerable. Por ello, hay que considerar inventar una buena función heurística: queremos encontrar soluciones cortas (pocos movimientos de las piezas) así que necesitamos una función heurística que **nunca sobre estime** el número de pasos hacia la meta. Mencionaremos dos posibles heurísticas:

1. $h_1 =$ el número de piezas que se encuentran en posiciones incorrectas respecto a la meta: para la Figura 2.2 (izquierda) todas las fichas (salvo la 7) están en desorden, por tanto $h_1 = 7$. h_1 es admisible por que cualquier ficha que esté en posición equivocada debe desplazarse al menos una vez.
2. $h_2 =$ la suma de las distancias de las fichas a sus posiciones meta, tal distancia (también llamada Distancia Manhattan) se cuenta en espacios vertical más horizontal (i.e., desplazamientos). h_2 es admisible debido a que cualquier movimiento sólo puede mover una ficha un paso más cercano hacia la meta: en la Figura 2.2 (izquierda), para las fichas consideradas en orden 1 a 8, $h_2 = 2 + 3 + 3 + 2 + 4 + 2 + 0 + 2$.

Una forma para caracterizar la calidad de una fución heurística es calcular el **factor de ramificación efectivo** b^*. Si el cual es el número total de nodos expandidos por A^* es m y la profundidad donde se encuantra una solución es d, entonces B^* es el factor de ramificación que un árbol con profundidad uniforme de d debería tener para contener m nodos:

$$m = 1 + b^* + (b^*)^2 + \ldots + (b^*)^d$$

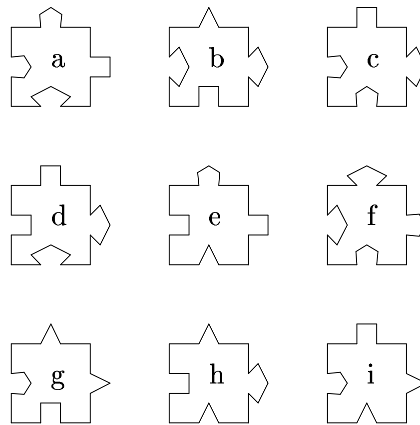
Por ejemplo, si A^* encuentra una solución a profundidad 5 utilizando 52 entonces $b^* = 1.91$. Dado que generalmente la heurística exhibe su factor b^* de manera constante, se puede medir expermientalmente ante una instancia pequeña de ejemplos. Una buena heurística debería tener un factor b^* cercano a 1. Experimentalmente, en [RN95] se reporta que los factores b^* para h_1 y h_2 son respectivamente 1.48 y 1.26, mostrando que h_2 es mejor que h_1. A partir de las definiciones de las heurísticas, se cumple que para cualquier nodo n, $h_2(n) \geq h_1(n)$. Decimos entonces que h_2 domina a h_1, lo cual tiene repercusión en la eficiencia: A^* utilizando a h_2 expanderá menos nodos en promedio que si utilizara a h_1. La moraleja es que si debemos elegir entre dos (o más) heurísticas, *es mejor utilizar aquella con valor mayor* pero que no sobre estime el costo real.

Podemos ingeniárnoslas para crear funciones heurísticas suponiendo que podemos eliminar algunas restricciones en el problema original: si las piezas del rompecabezas se pudiesen mover en cualquier dirección en vez de solo en la casilla vacía adyacente entonces claramente h_1 indicaría el número de pasos para la solución más corta. Similarmente, si se permitiese mover las fichas aún sobre casillas ocupadas, entonces h_2 daría la mejor solución.

Un subproblema obtenido con menos restricciones sobre los operadores de un problema original se denomina *problema relajado*. La heurística de cómo construír heurísticas es que **el costo de una solución exacta para un problema relajado es una buena heurística para el problema original**. Adicionalmente, la información estadística ofrece a menudo pistas a seguir de qué valores conviene considerar de acuerdo al comportamiento del problema. A menudo se puede elegir aspectos del estado del problema que contribuyen a la evaluación de la heurística, casi siempre en relación con las restricciones.

Tarea

Construya un programa Prolog para armar el rompezcabezas cuyas fichas se muestran en la Figura 2.3.2. Elija una estructura de datos adecuada para representar las piezas de tal forma que permita "rotar" y "empalmar" adecuadamente.



2.4 Algoritmos para Mejora Iterativa

Existen problemas que tienen la propiedad de que la descripción del estado contiene toda la información necesaria para la solución, ello tiene la consecuencia que la elección de una ruta es irrelevante para su solución (no importa el estado inicial). Algunos ejemplos de tales problemas son: las 8 reinas, arreglo de circuitos VLSI. El primero de ellos (toy problem) consiste en acomodar en un tablero de ajedrez a las ocho reinas de tal suerte que no se ataquen mutuamente. El segundo consiste en arreglar las compuertas en el diseño de un circuito VLSI de manera adecuada (minimizar el área ocupada y la longitud de las conexiones). En tales casos, los algoritmos de mejora iterativa ofrecen los mejores resultados, la idea general es que a partir de una configuración dada **realizar modificaciones para mejorar su calidad**.

Supongamos que podemos disponer de todos los estados del problema en una superficie: la altura en cualquier punto corresponde al valor de su función de evaluación. La idea de

la mejora iterativa es moverse sobre la superficie para alcanzar los picos que representan soluciones óptimas. Existen dos familias fundamentales de este tipo de algoritmos:

- **Ascenso a colina** (descenso de gradiente) siempre intentan aquel cambio que permite mejorar el estado actual.
- **Recocido simulado** realizan cambios (perturbaciones) que podrían empeorar temporalmente la situación.

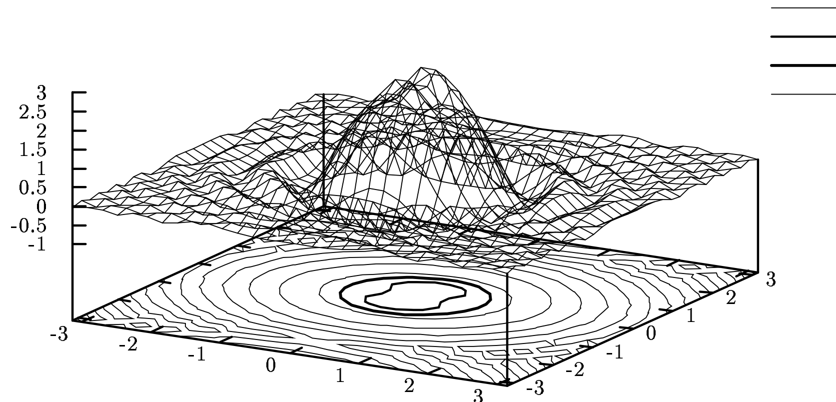


Figura 2.8: Los algoritmos de mejora iterativa intentan encontrar picos sobre la superficie de los estados del problema, la altura de un punto es el valor d ela función de evaluación.

Ascenso a colina

El algoritmo consiste esencialmente en un ciclo que se mueve en la dirección de valor creciente, lo único que se registra es el estado actual (un nodo ó partícula) y su evaluación (por tanto no almacena un árbol). Si hay más de un posible candidato (nodo sucesor), entonces se elige uno de manera aleatoria. Hay tres inconvenientes :

1. Máximo local: es un pico pero no el más grande del espacio de búsqueda, el algoritmo si se encuentra con un máximo local podría detenerse y regresar ese valor como solución (no óptima).
2. Plató: es un área plana, la búsqueda procede aleatoriamente.
3. Picos pronunciados: la búsqueda podría oscilar entre los lados de la superficie entre los picos progresando muy poco hacia la cima.

En cada caso, el algoritmo alcanza un punto en el cual queda atorado, lo mejor en estos casos es comenzar de nuevo en un punto diferente. Generalmente, una solución razonablemente buena puede encontrarse tras un número relativamente pequeño de iteraciones.

```
solution function acensoColina(problema) {
        nodo n1,n2;

        n1 = nodoInicial(problema);
        for(; ;) {
                n2 =  sucesor de mayor valor de n1
                if eval(n2) < eval(n1) return n1
                n1 = n2;
        }
```

Recocido simulado

Por su parte, en lugar de comenzar de nuevo en un punto aleatorio tras quedarse estancado, se puede preferir que la búsqueda descienda unos pasos para escapar del máximo local. El recocido simulado elige un candidato sucesor de manera aleatoria, si tal elección mejora la situación se continúa en esa dirección, de lo contrario realiza el movimiento con probabilidad menor a 1, i.e. la probabilidad decrece exponencialmente de acuerdo a la "maldad" del movimiento, representado por Δe. El algoritmo se muestra a continuación: T representa la probabilidad, a mayor valor se presenan movimientos "malos", mientras que cuando tiende a 0 son mejores.

El nombre de recocido simulado está inspirado en el proceso de forja en metales: se mete el metal en agua para enfriarlo gradualmente. La función `val` representa la energía total del material, T la temperatura, `sched` la tasa de enfriamiento.

```
solution function reCocidos(problema, sched) {
        /* sched es una funcion t → T */
        nodo n1,n2;
        time t;
        float T; /* temperatura */

        n1 = nodoInicial(problema);
        for(t=1; t<∞;t++) {
                T = sched(t);
                if T== 0  return n1;
                n2 =  sucesor aleatorio de n1;
                Δe = val(n2) - val(n1);
                if Δe > 0 n1 = n2;
                else n1 = n2 con probabilidad e^((Δe)/T);
        }
```

El recocido simulado ha sido utilizado desde los 80's para resolver problemas como el mencionado de diseño VLSI. Se ha utilizado con bastante éxito en problemas de optimización a gran escala.

Capítulo 3

Juegos

Los juegos han atrapado muchas veces a las facultades intelectuales humanas (algunas veces a grado alarmante) desde el comienzo de la civilización. Los juegos de mesa como el Ajedrez y Go presentan la ventaja de ofrecer un entorno de competencia abstracta, la cual hace que el proceso de jugar constituya un reto de investigación de la IA desde el inicio de esta disciplina. En 1950 Claude Shannon y Alan Turing escribieron los primeros programas para jugar ajedrez. Desde entonces, se han logrado muchos progresos hasta el nivel de retar a campeones mundiales humanos (e incluso, ganarles).

Generalmente, el estado de un juego es sencillo de representar y los agentes contendientes actúan dentro de los límites de un conjunto bien definido de reglas. Ello permite considerar al proceso de jugar como una idealización de mundos en los cuales los agentes antagónicos actúan con el propósito de obstaculizar y opacar al oponente. Recientemente, juegos más dinámicos (como el fut-bol) también han atraído la atención de los investigadores en IA[1] en particular, en lo concerniente a establecer protocolos de cooperación, negociación, etc.

El Ajedrez se eligió históricamente por varias razones:

- Un programa que pueda jugar al Ajedrez es una prueba de existencia de una máquina que está realizando un proceso que requiere inteligencia.
- La simplicidad de las reglas y el hecho que el entorno sea completamente accedible significa que se puede representar al juego como una búsqueda através del espacio de las posiciones posibles en el tablero.
- La presencia del agente antagónico introduce incerteza (contingencia) en el juego, ya que en general no se sabe con certeza qué hará a continuación el oponente. Ello hace que el problema de decidir qué hacer sea más complicado que como se consideró en las técnicas de búsqueda descritas en el capítulo anterior. Nótese que la incertidumbre no es aleatoriedad (como soltar un dado), sino que el oponente intentará realizar alguna acción con efectos daniños a las metas del contrario.

Pero lo que hace diferente al área de jugar juegos es que en general son difíciles de resolver. Por ejemplo, en el Ajedrez se tiene un factor promedio de ramificación de 35, en promedio cada jugador realiza 50 movimientos por lo que se tiene un espacio de 35^{100} nodos, aún cuando hay aproximadamente 10^{40} posiciones legales posibles distintas! La tremenda complejidad de los juegos introduce un tipo de incerteza: no se tiene el tiempo suficiente para calcular las consecuencias exactas de cualquier movimiento. Por tanto, cada agente

[1] Periódicamente se lleva a cabo el campeonato *RoboCup*

contendiente debe suponer (intuir) lo mejor posible en base a su experiencia y actuar antes de que se tenga la total certeza de qué acción tomar. En este aspecto, jugar juegos es más cercano al mundo real.

Dado que se habitúa jugar dentro de límites temporales, en los juegos se penaliza la ineficiencia. Por tanto, el área de investigación en juegos[2] ha creado una serie de ideas interesantes que permiten hacer el mejor uso del tiempo para tomar buenas decisiones, cuando alcanzar decisiones óptimas no es posible.

Primero se discutirá cómo realizar la mejor decisión teórica y entonces se analizarán las técnicas que permiten elegir un buen movimiento cuando el tiempo es limitado. El proceso de *poda* permite eliminar ramas del árbol de búsqueda que no aportan al resultado final, y las funciones heurísticas de evaluación permitirán aproximarse al valor de utilidad real de un estado sin necesidad de realizar una búsqueda exhaustiva.

3.1 Desiciones perfectas (en Juegos de dos participantes)

Consideremos el caso de dos contendientes que los llamaremos MAX y MIN. Las jugadas son alternadas, MAX mueve primero. Al final del juego se premia al ganador (o penaliza al perdedor).

Formalmente, un juego se define como un problema de búsqueda con los siguientes componentes:

- **Estado inicial**, la posición del tablero y a quien le toca comenzar.
- El conjunto de **operadores** que definen los movimientos legales de los participantes.
- Una **prueba terminal** para indicar cuando ha terminado el juego.
- Una **función de utilidad** que da el valor (generalmente numérico) resultado del juego: *ganar, perder, empatar.*

Si nos ponemos en el lugar del agente MAX, debemos encontrar una **estrategia** que permita alcanzar un estado terminal con el resultado de ganar sin importar los movimientos que el oponente MIN haga: la estrategia incluye cada respuesta correcta para MAX ante cada jugada de MIN.

Discutiremos cómo encontrar la mejor estrategia racional, aún cuando en la práctica no tengamos el tiempo suficiente para encontrarla. Para ello, haremos uso de un juego extremadamente simple (pues juegos tan triviales de resolver como el *gato*, su árbol de estados correspondiente es muy grande) que se muestra en la Figura 3.1. Los movimientos posibles para MAX están etiquetados por A_1, A_2, A_3 mientras que las respuestas de MIN para A_1 son $A_{1,1}, A_{1,2}$, etc. Decimos que éste árbol tiene profundidad de *un movimiento* ødos turnos, cada turno se denomina también *ply.* Se muestran en los nodos hoja las utilidades del juego.

Algoritmo *minimax*

Este algoritmo está diseñado para determinar la estrategia óptima para MAX y por tanto cual es el mejor movimiento inicial. Consta de 5 pasos:

1. Generar el árbol completo,

[2] De aquí en adelante, nos referiremos por “juegos” conjuntamente al proceso de jugar y a la descripción del juego mismo.

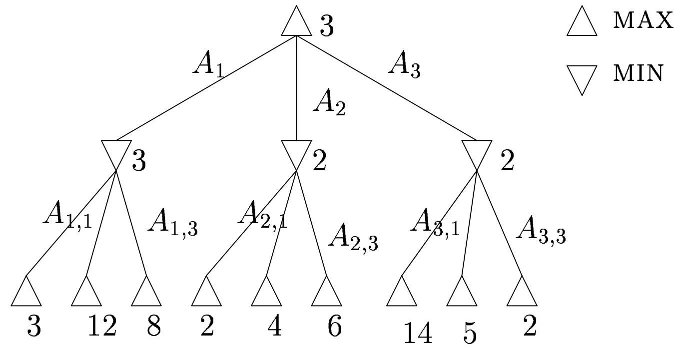


Figura 3.1: Un juego de 2 turnos generado por el algoritmo *minimax*. El mejor movimiento de MAX es A_1, la mejor respuesta de MIN es $A_{1,1}$

2. Aplicar la función de utilidad a cada estado terminal (hoja del árbol).
3. Utilizar la utilidad de los estados terminales para determinar la utilidad de los nodos en el nivel superior inmediato.
4. Propagar los valores de utilidad hacia los nodos en cada nivel superior, hasta alcanzar la raíz del árbol.
5. En la raíz, MAX elige el movimiento que le conduce al valor mayor.

```
operator function MinimaxD(juego) {
        /* Selecciona los movimientos a ser evaluados*/
        foreach op in juego.operadores do
             op.valor = minimaxV(juego, aplica(op, juego));
        end
        return el operador op con mayor valor
}

valor function minimaxD(juego, estado) {
        if terminal(estado) return estado.utilidad;
        else if juego.turno == MAX
             return mayor valor de minimaxV( succ(estado), juego);
        else
             return menor valor de minimaxV( succ(estado), juego);
}
```

La función `minimaxD` selecciona los movimientos posibles, que son evaluados por la función `minimaxV`. Si la profundidad del árbol de m y existen b movimientos legales en cada punto, entonces la complejidad en tiempo de minimax es $O(b^m)$, sus requerimientos en espacio son lineales en m y b (dado que utiliza recursión en lugar de almacenar listas de nodos). Evidentemente, para juegos reales los requerimientos de tiempo y espacio son prohibitivos, sin embargo éste algoritmo sirve como base para métodos más realistas así como para el análisis matemático de los juegos.

EL algoritmo minimax asume que se tiene tiempo suficiente para buscar sobre el árbol completo, lo cual es impráctico. En su artículo original, Shannon propuso que en vez de aplicar la función de evaluación, el programa debería acortar la búsqueda y aplicar una función de evaluación heurística a las hojas del árbol. De esa manera, el algoritmo minimax se modifica en

1. Reemplazar la función de utilidad por una función heurística EVAL,
2. la prueba terminal en las hojas se reemplaza por una prueba de corte (poda).

La función de evaluación heurística da un valor *estimado* de la utilidad esperada del juego en una posición determinada: el desempeño del programa para jugar juegos dependerá extremadamente de la calidad de dicha función. Si no es acertada, guiará al programa hacia posiciones que son aparentemente "buenas" pero que en realidad son desastrozas.

- La función EVAL debe coincidir con el valor de la función de utilidad en las hojas (terminales).
- Debe ser muy breve!
- Debe reflejar las oportunidades reales de ganar.

El enfoque más directo para acortar el espacio de búsqueda es fijar un límite en la profundidad del árbol d, de tal suerte que todo nodo en ó bajo la profunidad d dan positivo en la prueba de corte.

3.2 Poda $\alpha - \beta$

3.3 El mundo del *Wumpus*

Capítulo 4

Representación del Conocimiento e Inferencia

Capítulo 5

Planificación

Capítulo 6

Aprendizaje Automático

Bibliografía

[NRJ98] Michael Wooldridge Nicholas R. Jennings, Katia Sycara. A Roadmap of Agent Research and Development. *Autonomous Agents and Multiagent Systems*, 1998.

[RN95] Stuart Russell and Peter Norvig. *Artificial Intelligence, a Modern Approach.* Prentice Hall, 1995.

[ST99] Fariba Sadri and Francesca Toni. Computational Logic and Multi-Agent Systems: a Roadmap. Technical report, Dept. of computing, Imperial College, 1999.