12
2Ej

UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

FACULTAD DE CONTADURIA Y ADMINISTRACION

C + + INTELIGENTE PARA SISTEMAS INTELIGENTES

SEMINARIO DE INVESTIGACION

I N F O R M A T I C A

QUE PARA OBTENER EL TITULO DE:

LICENCIADO EN INFORMATICA

P R E S E N T A :

MARIA DEL CARMEN EDNA MARQUEZ MARQUEZ

ASESOR DEL SEMINARIO: L.C. Y M.C.C. MARINA TORIZ GARCIA

MEXICO, D.F. 1996

TESIS CON
FALLA DE ORIGEN

VNIVERSIDAD NACIONAL
AVFNOMA DE
MEXICO

Ciudad Universitaria a 9 de Septiembre de 1996

Prentice Hall International
A Division Simon & Schuster International Group
Presente.

Como miembro del personal académico, adscrita a la Facultad de Contaduría y Administración de la Universidad Nacional Autónoma de México, en donde tengo el honor colaborar en la asesoría del Seminario de Investigación "Tesis", me es preocupación pertinente el fomentar la participación de los alumnos en el vasto quehacer de esta Magna Casa de Estudios.

La alumna María del Carmen Edna Márquez Márquez con número de cuenta 8500342-8 y promedio de 9.84, aspira a obtener el Título de Licenciado en Informática, dada su trayectoria académica (se anexan documentos) reúne los requisitos para optar por una mención honorífica. Consciente de su iniciativa, creatividad y ganas de hacer las cosas la exhorté a que también puede pasar a formar parte de aquellos QUE HACEN QUE PASEN LAS COSAS, con el fin de que continúe así y en un futuro no muy lejano sea líder de su país "MÉXICO", aceptando gustosamente el reto, mostrando y sacando a flote todo su empuje al obtener como producto de su Seminario de Investigación en Informática un sistema que es aplicable en todas las empresas del mundo tomando en cuenta que ningún lenguaje de programación orientada a objetos ha sido aún planeado para que pueda aplicarse directamente en la programación de sistemas inteligentes, así como un texto con las mismas características.

Por todo esto les presentamos los productos de dicho Seminario de Investigación esperando que sea de su interés y solicitando su valiosísimo apoyo para que sean publicados y comercializados, habiendo acordado las autoras que los fondos reunidos servirán para fundar el "REI" RECINTO EDUCATIVO INTELIGENTE de la Universidad Nacional Autónoma de México.

Estamos seguras de que todo esto servirá para fortalecer la imagen de la Universidad Nacional Autónoma de México como Máxima Casa de Estudios, marcando un hecho sin precedente y que también será estímulo a los demás estudiantes para que sigan encontrando en la UNAM el ente social, humanístico y científico para desarrollar sus capacidades.

Sin más por el momentb agradezco la atención que se sirva prestar a la presente.

ATENTAMENTE

MARINA TORRES GARCIA
Asesora del Seminario de Investigación.

OCT. 7 1996

SECRETARIA DE RELACIONES
Y EXTENSION UNIVERSITARIA

c c p. Lic Ma Guadalupe Torres Solís Secretaria de Personal Docente y Exámenes Profesionales
c c p L A José Luis Ayala Chacón Secretario de Relaciones y Extensión Universitaria.

UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

Ciudad Universitaria a 9 de Septiembre de 1996

Microsoft Corporation
Presente.

Como miembro del personal académico, adscrita a la Facultad de Contaduría y Administración de la Universidad Nacional Autónoma de México, en donde tengo el honor colaborar en la asesoría del Seminario de Investigación "Tesis", me es preocupación pertinente el fomentar la participación de los alumnos en el vasto quehacer de esta Magna Casa de Estudios.

La alumna María del Carmen Edna Márquez Márquez con número de cuenta 8500342-8 y promedio de 9.84, aspira a obtener el Título de Licenciado en Informática, dada su trayectoria académica (se anexan documentos) reúne los requisitos para optar por una mención honorífica. Consciente de su inicialiva, creatividad y ganas de hacer las cosas la exhorté a que también **puede pasar a formar parte de aquellos QUE HACEN QUE PASEN LAS COSAS**, con el fin de que continúe así y en un futuro no muy lejano sea líder de su país "MÉXICO", aceptando gustosamente el reto, mostrando y sacando a flote todo su empuje al obtener como producto de su Seminario de Investigación en Informática un sistema que es aplicable en todas las empresas del mundo tomando en cuenta que ningún lenguaje de programación orientada a objetos ha sido aún planeado para que pueda aplicarse directamente en la programación de sistemas inteligentes, así como un texto con las mismas características.

Por todo esto les presentamos los productos de dicho Seminario de Investigación esperando que sea de su interés y solicitando su valiosísimo apoyo para que sean publicados y comercializados, habiendo acordado las autoras que los fondos reunidos servirán para fundar el "REI" RECINTO EDUCATIVO INTELIGENTE de la Universidad Nacional Autónoma de México.

Estamos seguras de que todo esto servirá para fortalecer la imagen de la Universidad Nacional Autónoma de México como Máxima Casa de Estudios, marcando un hecho sin precedente y que también será estímulo a los demás estudiantes para que sigan encontrando en la UNAM el ente social, humanístico y científico para desarrollar sus capacidades.

Sin más por el momento agradezco la atención que se sirva prestar a la presente.

ATENTAMENTE

MARINA TORRES GARCÍA.
Asesora del Seminario de Investigación.

OCT. 7 1996

SECRETARÍA DE RELACIONES Y EXTENSIÓN UNIVERSITARIA

Humberto Medina

c c p Lic Ma Guadalupe Torres Solís Secretaria de Personal Docente y Exámenes Profesionales
c c p L A José Luis Ayala Chacón Secretario de Relaciones y Extensión Universitaria

Agradecimientos

A mis padres:

Antonio Márquez Nájera

Ma. Lidia Márquez Hernández

Porque son unos padres ejemplares que han sabido guiar a mis hermanos y a mi por el camino de la superación, el amor y el respeto.

A mis hermanos:

César y Guadalupe

Porque me han brindado siempre su apoyo, su comprensión, su cariño y por todo lo que he aprendido de ambos.

A mi Universidad:

Porque a través del esfuerzo que hace la sociedad para la formación profesionista de una parte de ésta, me dio el honor de pertenecer a ese reducido grupo, otorgándome muchas de las satisfacciones más grandes de mi vida.

A mi asesora:

L.C. y M.C.C. Marina Toriz García

Porque gracias a su ayuda ha sido posible la culminación de esta primera etapa de preparación profesional.

Indice general.

Capítulo III. Búsquedas de solución de problemas en el diseño de Sistemas Inteligentes.

Capítulo IV. Programación Orientada a Objetos.

Capítulo V. Programación de las directivas de C++ Inteligente.

Introducción.

Como resultado de que el desarrollo de los **Sistemas Inteligentes** hasta ahora sólo se centra en un reducido grupo de personas dedicadas a la *informática*, por la forma complicada en que se presenta, surge el interés por realizar esta investigación denominada **C++ Inteligente para Sistemas Inteligentes.**

Por un lado esta investigación trata los temas de **Inteligencia Artificial** y la **Programación Orientada a Objetos**, los cuales representan caminos que han sido creados para hacer eficiente el funcionamiento de las computadoras a través del correcto desempeño del *software*, tratando de acabar con los problemas que se han presentado a lo largo de la historia de la computación, en cuanto al desarrollo de sistemas.

Aquí se presenta la unión de estas dos vertientes de la computación debido a que juntas pueden aportar mucho al desarrollo de sistemas, en este caso de **Sistemas Inteligentes**, dando mayor flexibilidad a su realización.

Los **Sistemas Inteligentes** tienen una parte integral de los elementos de la **Inteligencia Artificial**, por lo que se tratan las características, elementos y descripción en general de la **Inteligencia Artificial**, vinculándolos con el desarrollo de **Sistemas Inteligentes**. Y debido a que el producto de esta investigación está desarrollado de acuerdo con la **Programación Orientada a Objetos** también se incluye como uno de los temas principales.

Estructura del texto.

Para el cumplimiento de los objetivos de esta investigación el texto ha sido estructurado en 5 capítulos integrados de la siguiente manera:

En el capitulo primero, Fundamentos de **Sistemas Inteligentes**, se hace una semblanza de los principios de los **Sistemas Inteligentes**, en el cual se habla acerca de los aspectos que de manera general componen a la **Inteligencia Artificial**, ilustrando sobre sus conceptos principales, su evolución, su relación con otras disciplinas, y sus herramientas y desarrollos ya realizados.

En el capitulo segundo, El conocimiento como parte del análisis de **Sistemas Inteligentes**, se trata como su nombre lo indica, de uno de los elementos principales dentro de los **Sistemas Inteligentes**, el *conocimiento*, a partir del que se desarrolla también la **Inteligencia Artificial**, y del que se han creado diversas maneras de representarlo y posteriormente utilizarlo.

En el capitulo tercero, Búsquedas de solución de problemas en el diseño de **Sistemas Inteligentes**, introduce a otro elemento de los **Sistemas Inteligentes**, a través del que se hará uso del *conocimiento* representado con el fin de dar solución a los problemas planteados acerca de tal *conocimiento*, de la mejor manera posible, este elemento son las búsquedas de solución de problemas.

En el capitulo cuarto, **Programación Orientada a Objetos**, se muestran los principios de la **Programación Orientada a Objetos**, aspectos sobre sus lenguajes principales y la forma en que se relaciona con la **Inteligencia Artificial**, comentando algunas de las herramientas creadas en su combinación.

Finalmente en el capitulo quinto, Programación de las directivas de C++ Inteligente, se encuentra el desarrollo de los archivos de cabecera .H (#include), para crear el producto C++ Inteligente, en el que se combina la **Inteligencia Artificial** y la **Programación Orientada a Objetos**, en pro de un desarrollo de **Sistemas Inteligentes** más sencillo pero más eficiente.

Resumen de la investigación.

Título.

C++ Inteligente para Sistemas Inteligentes.

Temas principales.

Inteligencia Artificial.
Programación Orientada a Objetos.
Sistemas Inteligentes.

Objetivo general.

Crear los archivos de cabecera .H (#include), también denominados directivas, del lenguaje C++ Inteligente, utilizando la **Programación Orientada a Objetos** y las bases de la **Inteligencia Artificial** para su aplicación en el desarrollo de **Sistemas Inteligentes.**

Objetivos específicos.

- Mostrar la posibilidad de realizar **Sistemas Inteligentes** de un modo más sencillo.
- Mostrar de manera clara y sencilla los aspectos principales que involucra la **Inteligencia Artificial.**
- Indicar las perspectivas surgidas en las ciencias de la computación como resultado del desarrollo de la **Inteligencia Artificial.**
- Introducir a más personas al desarrollo de **Sistemas Inteligentes** por medio de C++ Inteligente.
- Facilitar el aprendizaje de la **Inteligencia Artificial.**

- Mantener la eficiencia y transportabilidad de C++.
- Hacer compatible a C++ Inteligente con las versiones anteriores de C++.
- Actualizar a C++ al paradigma de **Sistemas Inteligentes** para producir *software* de calidad.
- Promover la ampliación del lenguaje C++ hacia el lenguaje C++ Inteligente.
- Poner al alcance de los desarrolladores de **Sistemas Inteligentes** las directivas (#include) desarrolladas a través de los productos del lenguaje C++ de Microsoft.

Antecedentes.

La **Programación Orientada a Objetos** comenzó a aplicarse a la **Inteligencia Artificial** a principios de la presente década, con el objeto de aprovechar las características principales que ofrece este *paradigma* y facilitar sus desarrollos. Sin embargo, se trata todavía de los lenguajes puros de la **Inteligencia Artificial**, como lo son *Lisp* y *Prolog*, sus versiones orientadas a objetos, como Prolog++, LOOPS (Lisp Object-Oriented Programming System), CLOS (Common Lisp Object System), entre otras. En todos estos casos se trata de lenguajes que están en los centros de investigación.

Ningún lenguaje de **Programación Orientada a Objetos** ha sido aún planeado para que pueda aplicarse directamente en la programación de **Sistemas Inteligentes.**

Situación actual de los temas.

El manejo tanto de la **Inteligencia Artificial** como de la **Programación Orientada a Objetos** se ha visto incrementado, como resultado de las ventajas que proporcionan para el desarrollo, implantación y mantenimiento de los sistemas.

Su utilización de manera independiente ha contribuido a realizar diversas aplicaciones en todo tipo de organización obteniendo resultados muy favorables, sin embargo, ha surgido la idea de

combinarlas para aprovechar de manera conjunta las principales características de cada una, por lo que se han creado diversas herramientas que pretenden alcanzar dicho objetivo.

Las herramientas creadas hasta ahora están basadas particularmente en lenguajes de **Inteligencia Artificial**, *Prolog* y *Lisp*, a los que se les han agregado aspectos de la **Programación Orientada a Objetos**, dichas herramientas debido a su complejidad no están al alcance de todos los desarrolladores de sistemas, puesto que pocos conocen el manejo de los lenguajes de **Inteligencia Artificial**.

Los **Sistemas Inteligentes** están hechos con lenguajes puros de **Inteligencia Artificial**, lo que hace poco común su desarrollo, por las dificultades que ésto implica tales como encontrar gente que conozca el uso de los lenguajes de **Inteligencia Artificial**, máquinas que soporten sus desarrollos y personas que puedan darles el mantenimiento necesario.

Justificación de la investigación.

Las herramientas hechas para **Inteligencia Artificial** con **Programación Orientada a Objetos** no son accesibles para todo tipo de programador y menos para las personas en común como se había planteado en su origen, para su utilización se requieren grandes conocimientos del área de **Inteligencia Artificial** y no resultan tan sencillos como pretendían sus creadores, existe también una gran diversidad de herramientas creadas sin un estándar definido, dificultando más su utilización. Por lo que tales herramientas no pueden ser utilizadas por la mayoría de los desarrolladores de **Sistemas Inteligentes**.

Con esta investigación se da a conocer a un mayor número de personas, especialmente a los que ya están familiarizados con el lenguaje C++ para que puedan utilizar el nuevo C++ inteligente y apliquen los principios de la **Inteligencia Artificial** dentro de los sistemas que desarrollen y logren obtener **Sistemas Inteligentes**. Dando una explicación lo más comprensible posible de las bases generales de los **Sistemas Inteligentes** tomada de la **Inteligencia Artificial**, así como de la **Programación Orientada a Objetos**, se aprovechan las

ventajas de la **Inteligencia Artificial** en conjunción con las de la **Programación Orientada a Objetos**, para introducir al máximo de personas relacionadas con la computación en ambos *paradigmas* y beneficiarse con el uso de sus principales características para el desarrollo de **Sistemas Inteligentes**.

La programación de los **Sistemas Inteligentes** será más sencilla, sin requerir de grandes o especializados equipos de cómputo si se realizan con un lenguaje como C+ inteligente que goza de gran portabilidad y facilidad de instalación de los sistemas desarrollados a través de este lenguaje.

Por otro lado, debido a que las máquinas cada vez tienen más potencial en su funcionamiento y las computadoras de la quinta generación están pensadas para trabajar con desarrollos de Inteligencia Artificial, es importante que la gente dedicada a la *informática* tenga conocimientos en el área para que pueda aprovechar al máximo los recursos de que disponga y logre desarrollar **Sistemas Inteligentes**. A través del uso de los archivos de cabecera .H (#include), pueden utilizarse los principios de la **Inteligencia Artificial**, para lograr este objetivo.

Problema.

No existe dentro de ningún lenguaje de **Programación Orientada a Objetos** las bases para la creación de sistemas que tengan un comportamiento inteligente.

Los lenguajes hechos en base a **Inteligencia Artificial** y **Programación Orientada a Objetos** no están al alcance de la mayoría de los desarrolladores de *software*, debido a que están basadas en lenguajes puros de **Inteligencia Artificial**.

Solución al problema.

Se han elaborado las *clases* necesarias que integran las directivas de #include para la extensión de un lenguaje ampliamente utilizado, obteniendo C++ inteligente, las que manejan las características de las búsquedas realizadas para las aplicaciones de **Sistemas Inteligentes.**

El producto final de esta investigación ya se ha enviado a Microsoft Corporation, para su inclusión dentro de las librerías con que cuenta el lenguaje C++ y generar un nuevo producto al cual le hemos denominado C++ Inteligente, para generalizar su uso y hacer más poderoso al mismo lenguaje de programación.

Hipótesis.

El uso del lenguaje C++ Inteligente facilita el desarrollo de **Sistemas Inteligentes.**

Aportación al tema.

Creación de las directivas (#include), para C++ Inteligente que manejan parte de los fundamentos de **Inteligencia Artificial** logrando introducir a un mayor número de desarrolladores de *software* a la misma por medio de un lenguaje ampliamente conocido, generalizando el desarrollo de **Sistemas Inteligentes.**

Dar a conocer a través del texto los principios que integran la **Inteligencia Artificial** y la **Programación Orientada a Objetos** bases para el desarrollo de **Sistemas Inteligentes,** en forma práctica, sencilla, y a través de casos contable-administrativos.

Alcance del trabajo.

A través de la inclusión de los archivos de cabecera (#include) en el lenguaje C++ Inteligente, todas las personas que lo utilicen podrán desarrollar **Sistemas Inteligentes** con mayor facilidad, ya que si no lo desean, no necesitan enterarse como funcionan estas directivas sino con el hecho de invocarlas obtendrán los resultados.

Relación con otros sistemas.

Está estrechamente ligado con el lenguaje C++, especialmente con las versiones de Microsoft. Con los **Sistemas Inteligentes** ya existentes.

Recopilación de la información.

Para la realización de esta investigación se ha utilizado información procedente de libros, revistas y diccionarios especializados en **Programación Orientada a Objetos, Inteligencia Artificial y Sistemas Inteligentes**, así como en documentos publicados en Internet por universidades y centros de investigación ocupados de estos temas.

Se acudió a las bibliotecas:
De la Facultad de Contaduría y Administración (FCA).
De la Dirección General de Servicios de Cómputo Académico (DGSCA).
Del Instituto de Investigaciones de Matemáticas Aplicadas y Sistemas (IIMAS).
Del Instituto de Investigaciones Biomédicas (IIB).
Central de la Universidad Nacional Autónoma de México.
De la Facultad de Ingeniería.
De la Facultad de Ciencias.
Del Centro de Información Científica y Humanística (CICH).

Enfoque.

Es teórico - práctico. Ya que los primeros 4 capítulos se componen del marco teórico y el último de un desarrollo práctico.

Universo.

Esta investigación está concebida para todos aquellos desarrolladores de sistemas que estén interesados en lograr una mayor calidad de sus productos y crear **Sistemas Inteligentes**.

Ubicación geográfica.

Actualmente el producto está disponible en un área local, que corresponde sólo a la UNAM y a Microsoft de México, pero posteriormente con su publicación y comercialización podrá ser internacional.

Espacio.

Pasado - Existen investigaciones acerca de Inteligencia Artificial y de la Programación Orientada a Objetos.
Presente - No existen desarrollos referentes a un C++ Inteligente.
Futuro - Se pondrá a disposición de los desarrolladores de sistemas un nuevo lenguaje inteligente.

Políticas de edición.

Para mejor comprensión de la investigación presentada por medio de este texto, cabe mencionar las políticas de edición que se tomaron en cuenta:

Los temas centrales de la investigación son resaltados con letras oscuras, así se encuentra: **Inteligencia Artificial, Programación Orientada a Objetos y Sistemas Inteligentes.**

Las términos utilizados a lo largo de la explicación que fueron incluidos en el glosario, se escriben con letra *cursiva*, con excepción de los incluidos dentro de títulos y subtítulos, así como en el nombre de las figuras y cuadros.

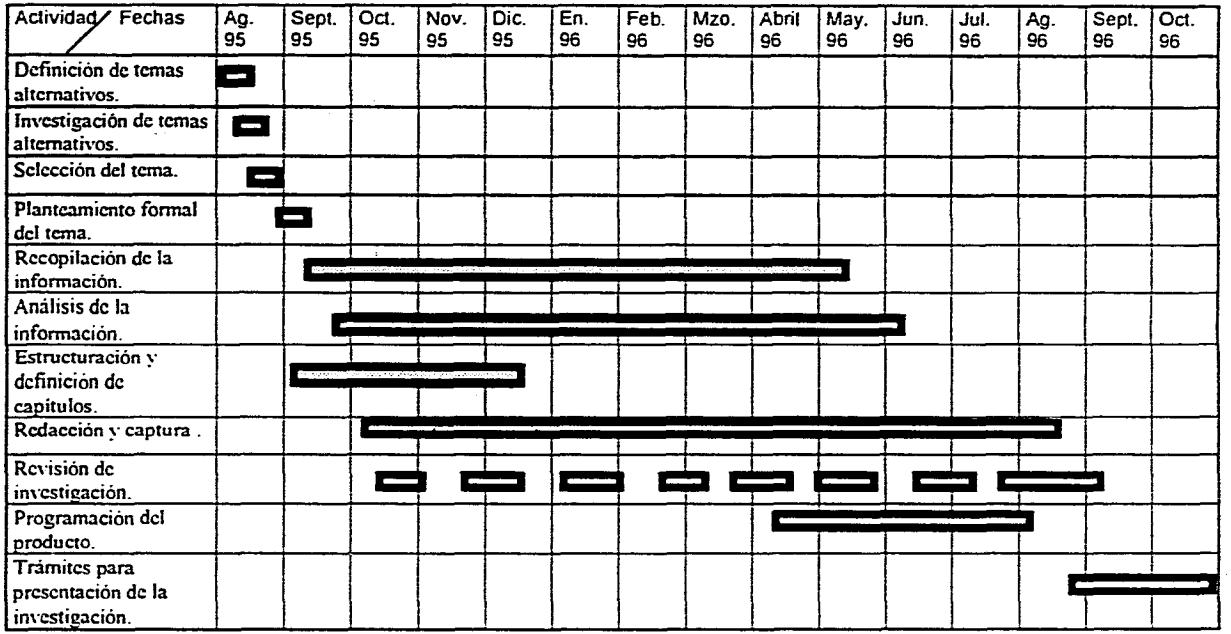
Las referencias bibliográficas se muestran colocando entre corchetes [ ] el apellido del autor y el año de publicación de la obra.

Los pies de página se identifican por números y tiene un tamaño de letra menor que el resto del texto.

Cuando se da un concepto dado por un autor se muestra entre comillas " " y el tamaño de la letra es menor.

Algunos términos son manejados no sólo en español sino también en inglés debido a que se les conoce en este idioma.

Gráfica de Gantt para la investigación, de Agosto de 1995 a Octubre de 1996.



Capítulo I.

Fundamentos de Sistemas Inteligentes.

Capítulo I . Fundamentos de Sistemas Inteligentes.

1. Introducción.

La historia de la computación está estrechamente relacionada con la necesidad del hombre de hacer su vida más sencilla, de manera que las máquinas puedan sustituirle en lo posible en las tareas que para él resulten riesgosas, repetitivas, de gran precisión y/o laboriosas, por esta razón se ha tratado de asemejar el funcionamiento de las computadoras con el comportamiento humano.

Como resultado del desarrollo que la computación ha tenido en esta búsqueda, surgió la **Inteligencia Artificial**, que si bien no iguala aún el comportamiento humano, tanto en el aspecto mental como físico, si se han obtenido importantes logros.

Por otro lado, la computadora es un medio que ha sido muy eficaz para la simulación y comprobación de los estudios, modelos y teorías surgidas acerca de los mecanismos de la inteligencia humana, lo que ha contribuido mucho al desarrollo de la **Inteligencia Artificial**.

Los **Sistemas Inteligentes**, tienen una parte integrante basada en la **Inteligencia Artificial** por lo que en este capítulo se incluyen aspectos generales de la misma.

2. Concepto de Sistemas Inteligentes.

Hablar de **Sistemas Inteligentes** es referirse a la **Inteligencia Artificial**, en cuanto que toma parte de ésta para integrarse, por tal motivo es necesario definir también a esta rama de la computación.

Algunas de las definiciones dadas por los autores a la Inteligencia Artificial son:

Para Edward A. Feigenbaum la Intelligencia Artificial es:

" La parte de las ciencias de la computación interesada en el diseño de sistemas de computación inteligentes, es decir, sistemas que muestren las características asociadas con la inteligencia humana como comprensión del lenguaje, aprendizaje, razonamiento y resolución de problemas". [Feigenbaum 89].

Marvin Minsky la define como:

" La ciencia que hace que las máquinas hagan cosas que requerirían inteligencia si fueran hechas por un hombre". [Turban 92].

Rauch afirma que la Intelligencia Artificial es :

" Una técnica de *software* que los programas utilizan para resolver problemas expresados en términos simbólicos más que numéricos ". [Rauch 89].

El diccionario Microsoft Press la define como:

" La rama de las ciencias de la computación que trata con las computadoras habilitadas para emular aspectos de la inteligencia como el reconocimiento de voz, la *deducción*, la *inferencia*, la creatividad de respuesta, la habilidad para aprender de la experiencia obtenida y la habilidad de hacer *inferencias* razonables a partir de información incompleta". [Micros 91].

Por su lado, los vocablos que conforman el término Inteligencia Artificial se definen como :

Inteligencia, se refiere a la facultad del hombre de conocer y comprender, la capacidad de adaptación a situaciones nuevas, empleando los recursos del pensamiento, así como el manejo de relaciones y *símbolos abstractos*. [Salvat 78].

Inteligencia es la acción de comprender, la capacidad de saber o aprender. [D.Univ.72].

Inteligencia es la facultad de comprender y conocer. [Larousse 80].

Todos los autores coinciden en definir a la palabra Artificial como:
Lo que no es natural, sino hecho por el hombre o lo hecho por la mano o arte del hombre.

Uniendo las definiciones de ambos términos puede decirse que la **Inteligencia Artificial** se ocupa de representar con medios creados por el hombre, en este caso las computadoras, los aspectos determinantes del comportamiento inteligente del hombre.

Sistema es un conjunto de elementos relacionados entre si que operan con vistas a un mismo objetivo.

Dentro de la computación un sistema se integra por el conjunto de procedimientos, documentos, métodos, datos, equipo físico y personal que realizan juntos una compleja serie de operaciones referentes a un dominio específico.

De acuerdo a la definición de sistemas y a la de **Inteligencia Artificial**, de donde se deriva el adjetivo de inteligente finalmente diremos que los **Sistemas Inteligentes** son el conjunto de elementos interrelacionados, en donde se combina la tecnología, los conocimientos y métodos de inferencia propios de la **Inteligencia Artificial** que tienen como objetivo común el realizar acciones que implican un comportamiento inteligente.

Los **Sistemas Inteligentes** son de gran utilidad como apoyo para la toma de decisiones, debido a que contienen hechos que pueden ser tanto cualitativos como cuantitativos, los que pueden llegar a cambiar con frecuencia, teniendo que anticiparse a circunstancias, considerando la incertidumbre y contando con la experiencia y conocimientos de personas expertas en la toma de decisiones.

3. Diferencias de la Inteligencia Artificial con los demás campos de la computación.

Los aspectos clave que diferencian a la **Inteligencia Artificial** del resto de los campos de la computación en general son : el procesamiento de simbolos, la heuristica, el uso de la inferencia y el conocimiento.

3.1 Procesamiento simbólico.

Los problemas que se resuelven a través de la **Inteligencia Artificial** no son considerados como un conjunto de operaciones, ecuaciones o aspectos matemáticos de computación que utilizan sólo elementos numéricos, sino que estos problemas se manejan con *simbolos*.

Mediante los *simbolos* se pueden representar los conceptos involucrados en el problema, a los que se les aplican métodos y reglas para su manipulación. La **Inteligencia Artificial** representa el *conocimiento* como un conjunto de *simbolos*, que indican los conceptos del problema.

Los *simbolos* son entidades que se utilizan para representar personas, objetos, conceptos, operaciones, relaciones o los atributos de un objeto en el mundo real.

Un *simbolo* para la **Inteligencia Artificial** es definido como una cadena de caracteres, una variable o una constante que representan aspectos reales, los cuales pueden combinarse entre si y mostrar una interrelación entre ellos, llegando incluso a formar estructuras de *simbolos*, es decir, *estructuras simbólicas de datos*. La **Inteligencia Artificial** resuelve los problemas planteados a través de la manipulación de *simbolos*, lo que da lugar a la *representación del conocimiento*.

En cambio, los programas de cómputo tradicional utilizan sólo números, no *símbolos*. Originalmente las computadoras se diseñaron sólo para el procesamiento numérico, pero como el pensamiento humano es a través de *símbolos* y la Inteligencia Artificial se basa en el proceso mental humano se trata de asemejar esa actividad con el procesamiento de *símbolos*. Así que la manipulación de *símbolos* es una de las partes principales de la **Inteligencia Artificial**.

3.2 Heurística.

Es una característica esencial de la **Inteligencia Artificial**, representa a un conjunto de métodos para el procesamiento de la información. Los problemas en **Inteligencia Artificial** no se resuelven de una forma algoritmica como sucede en la computación común[1], puesto que, el proceso de razonamiento humano no siempre actúa de forma determinística, ya que no sigue una secuencia precisa de pasos.

3.3 Inferencia.

El razonamiento implica el utilizar la *inferencia*, que se define como el proceso por el cual es obtenida una conclusión basándose en una serie de *hechos* a través de métodos de *inducción* y *deducción*, es decir, se derivan nuevos *hechos* a partir de otros *hechos* o *conocimientos* previos.

La **Inteligencia Artificial** deriva hechos y conocimientos nuevos a partir de otros anteriores utilizando los métodos de *inferencia*.

[1] La computación convencional o común sigue para la solución de problemas un conjunto de pasos con una secuencia lógica, en el que se define correctamente un principio y un fin, se especifica paso a paso el uso que se dará a los datos, es definido claramente el *algoritmo* a seguir para dar solución a un problema.

A través del mecanismo inferencial se realiza la interpretación de los *conocimientos* almacenados en la base y se efectúan deducciones lógicas al igual que ciertas modificaciones en la *base de conocimientos* como resultado de los nuevos *conocimientos* obtenidos.

3.4 Conocimiento.

Existe una distinción entre lo que se entiende por datos, información y *conocimiento.*
Los datos se refieren a los signos, los números o las letras que por sí solos no tienen ningún significado.

La información es el conjunto de datos organizados, con lo que éstos adquieren un significado específico.

El *conocimiento,* por su parte, se refiere a la capacidad humana de entender, aprender, percibir, obtener experiencia y poder organizar la información que se tiene para dar solución a un problema. El *conocimiento* es más abstracto que la información o los datos aunque éstos son más numerosos.

Las computadoras por sí solas no pueden crear el *conocimiento,* por lo que hacen uso del *conocimiento* que previamente se les proporciona, pudiendo utilizarlo de manera semejante al hombre y adquirir más de manera posterior con base en el que ya poseen.

El *conocimiento* incluye *hechos,* conceptos, teorías, métodos de *inferencia, heurística,* procedimientos y relaciones, todo un conjunto de información organizada que se utilizará para dar solución a un problema.

El *conocimiento* es información simbólica, *hechos* y sus relaciones que se utilizan para dar solución a problemas.

El *conocimiento* relacionado en conjunto forma la *base de conocimientos*, que es de gran importancia para la Inteligencia Artificial, debido a que a dicha base se le aplican las técnicas de *inferencia* para llegar a la solución de un problema, además de tener la posibilidad de obtener otros *conocimientos*.

Para procesar el *conocimiento* se requieren operaciones que van más allá de las tradicionales de ordenamiento, selección, almacenamiento y cálculo, es necesario clasificar, tratar de formar conceptos, hacer abstracciones, razonar, planear, modelar, aprender y usar reglas de *inferencia* y *heurística*.
Para lograr todo eso se utilizan las reglas de *deducción* o de *inducción*.

3.4.1 Bases de conocimientos y bases de datos.

Ambos tipos de bases difieren significativamente por el tipo de información que pueden almacenar, el tipo de interrelación entre datos que pueden manejar, es decir, ya sean conocimientos específicos de la aplicación o conocimientos de sentido común, y la clase de adiestramiento que actualiza la información almacenada. [Rauch 89].

Las *bases de conocimiento* al igual que las bases de datos almacenan *hechos* definidos y directos. Pero además las *bases de conocimientos* almacenan elementos del tipo causa-efecto, reglas observadas o demostradas e información imprecisa, probabilística e indeterminada, lo que permite tener *conocimientos borrosos*.

En la base de datos sólo se almacena información precisa.

La *base de conocimientos* de manera simbólica representa *conocimientos* de *hechos* e información general, así como heurísticos, tales como juicios, intuición y experiencias sobre determinada área del saber.

En la *base de conocimientos* las secciones están separadas permitiendo así añadir nuevas secciones de *conocimiento* y lograr que crezca y se mantenga actualizada. En cuanto el manejo del sentido común las *bases de conocimientos* almacenan y manejan relaciones entre *hechos* y secciones de información muy complejas pues el objetivo es identificar y codificar la enorme variedad de relaciones existentes permitiendo desarrollar e inferir una gran cantidad de *conocimientos* causales y de sentido común sobre todas las posibilidades de una determinada área de aplicación. [Rauch 89].

Las diferencias entre ambos tipos de bases se resume en el siguiente cuadro.

	Bases de datos	**Base de conocimientos**
Contenido	Colección de datos hechos.	Alto nivel de abstracción Clases de objetos (hechos).
Complejidad	Los elementos almacenados son simples.	Los elementos almacenados son relaciones complejas.
Tiempo y dependencia	El dato cambia frecuentemente.	El conocimiento cambia con poca frecuencia.
Tamaño	Gran número de datos.	Diversas relaciones sobre clases de objetos (hechos).
Uso	Propósito operacional.	Análisis, planeación, etc..

Cuadro 1. Diferencias entre las bases de datos y de conocimientos. [Wah 90].

En general es posible establecer que la **Inteligencia Artificial** difiere de la computación convencional en que la información que maneja no se refiere únicamente a datos numéricos sino también simbólicos, por lo que se busca a su vez un razonamiento lógico y no sólo cálculos cuantitativos, además permite almacenar junto con los datos a los procedimientos y métodos que harán uso de esa información.

Como resultado de la manipulación simbólica, no utiliza las representaciones, manejo y obtención comunes de datos, sino que se crean otras que le permiten realizar la manipulación de *conocimientos* y de resultados.

No siempre utiliza *algoritmos* definidos para la solución de problemas, ya que se apoya en métodos heurísticos y de *inferencia*.

Los programas de Inteligencia **Artificial** se deben de adaptar a situaciones en las que la información que integra el *conocimiento*, no es exacta ni completa[2] .

De acuerdo con lo anterior los **Sistemas Inteligentes** tiene las siguientes características [Bielawski 91]:

- Los **Sistemas Inteligentes** se comportan lógicamente.
- Los **Sistemas Inteligentes** resuelven problemas complejos.
- Los **Sistemas Inteligentes** son sensibles y adaptables.
- Los **Sistemas Inteligentes** cuentan con una ruta no determinada dentro del programa.
- Los **Sistemas Inteligentes** hacen uso efectivo de la información existente.
- Los **Sistemas Inteligentes** tienen una forma sencilla de interactuar con el usuario.

[2] Los programas de **Inteligencia Artificial** tienen su punto fuerte en su capacidad para resolver problemas que implican *símbolos*, conceptos e ideas simbólicas, en contraste con los programas informáticos convencionales que efectúan operaciones aritméticas. Han sido diseñados para comprender la relación entre conceptos e ideas. [Rauch 89].

4. Desarrollo histórico de la Inteligencia Artificial.

El nombre de **Inteligencia Artificial** fue acordado en el año de 1956 en una conferencia dada en Dattmouth, Estados Unidos, por Jonh McCarthy en la que participaron también M. Minsky, C. Shannon, H. Simon y A. Newell, ellos acordaron la posibilidad de hacer programas de computadora que actuaran de manera inteligente.

Sin embargo, su historia no comenzó allí, ya en los años 30's Alan Turing había hecho el primer diseño de **Inteligencia Artificial**, era un juego conocido como " la *prueba de Turing*" o "el juego de la imitación"; el que consistía básicemente en que una persona se debía comunicar con un incógnito a través de preguntas y respuestas en mensajes, teniendo al

final que adivinar si el incógnito era otra persone o se trataba de una computadora, el juego implicaba que la computedora entendiera el *lenguaje natural* y tuviera la capacidad de razonamiento, con ésto edvirtió que para lograrlo se requería del procesamiento de *simbolos* y no sólo de números. Por haber sido el primer diseñador de algo que implicara un comportamiento inteligente de las computadoras se le denominó a Alan Turing como el padre de la **Inteligencia Artificial**.

En 1957 Newell, Shaw y Simon muestran un programa del juego de ajedrez basado en un método sugerido antes por Shannon, en el que colaboran personas expertas en ajedrez y psicólogos para su desarrollo, y se crea un lenguaje especial para la manipulación de símbolos lógicos por computadora e través de apuntadores y ligas, llamado IPL-1 (Information Processing Language 1), precursor del lenguaje *Lisp* (List Processing Language).

El primer programa de **Inteligencia Artificial** presentedo fue Logist Theorist, un programa demostrador de teoremes lógicos, trabajaba con *lógica proposicional* usando *heurística*.

En 1958 desarrollaron el programa GPS (General Problem Solver), que manejaba búsquedas generales para la solución de problemas, analizaba las situaciones presentadas y el objetivo a alcanzar, podía utilizarse en la solución de problemas de lógica elemental, juegos de ajedrez, acertijos, expresiones algebraicas y sistemas de preguntas y respuestas.

En 1958 Jonh McCarthy desarrolló el lenguaje *Lisp*, uno de los más utilizados actualmente para la implantación de aplicaciones de Inteligencia Artificial.

A fines de la década de los 50's comienzan a darse a conocer los primeros reportes acerca del trabajo realizado en este campo de la computación con lo que surgen más personas interesadas en su investigación.

En los años 60's la Inteligencia Artificial ya establecida como disciplina, marca su verdadero despegue, en diversas universidades e institutos de Estados Unidos y Europa se realizaba ya investigación acerca de esta nueva área de la computación. Se crea el primer *sistema experto* por Joshua Lederbeg, Bruce Buchanan y Edward Fiegenbaum en Stanford, denominado DENDRAL, utilizado en el área de química y con el que demuestran que los *sistemas expertos* requieren una gran *base de conocimientos*.

En 1962 G. W. Ernst crea el primer *robot* controlado por computadora, se trataba de un brazo mecánico con hombro, codo y una tenaza.

J. Robinson desarrolla en 1965 un nuevo tipo de procesamiento lógico basado en la manipulación sistemática de reducción a lo absurdo, que hace posible expresar de manera formal y ser manipulados por la máquina a nuevos problemas, este es el punto de partida del lenguaje *Prolog*.

A finales de los 60's Marvin Minsky y S. Papert comienzan a realizar estudios sobre *visión por computadora*.

A principios de los 70's el interés por la Inteligencia Artificial crece, Terry Winograd realiza el proyecto SHDRUL, que es un *robot* conducido a través de instrucciones en inglés para manipular cubos de colores.

En 1971 se desarrolla el lenguaje *Prolog* en Francia por Colmerauer.

Para finales de esta década ya existían trabajos en las áreas de percepción y entendimiento del *lenguaje natural*, *visión por computadora* y *sistemas expertos*; los que se aplicaban en disciplinas como la medicina, química, psicología, matemáticas y en finanzas. A pesar de ello en esta década la **Inteligencia Artificial** tuvo grandes problemas resultado del recorte presupuestal del que fue objeto debido a las promesas exageradas no alcanzadas que se habían hecho en torno a sus aplicaciones, provocando gran incredulidad de los organismos que financiaban su investigación en América del Norte y Europa.

En la década de los años 80's se inicia la comercialización de las aplicaciones de **Inteligencia Artificial**, multiplicándose sus productos, se diseñan herramientas especiales tanto en *hardware* como en *software* para el desarrollo de sus aplicaciones, creciendo el interés por parte de muchas compañías particulares y no sólo de universidades para su investigación, retornando la confianza antes perdida al campo de la **Inteligencia Artificial.**

Los japoneses prometieron para esta década de los 90's la creación de una quinta generación de computadoras, la cual se refiere a desarrollos de **Inteligencia Artificial**, estas máquinas debían tener la capacidad de entender el *lenguaje natural*, reconocer imágenes, tomar decisiones inteligentes, resolver problemas sociales y tener grandes *bases de conocimientos*, entre otras posibilidades, sin embargo, aún no ha sido posible obtener por completo dichas características en las máquinas.

4.1 Inicios de la Inteligencia Artificial en México.

En México se introdujo la computación a finales de los años 50's, cuando se instaló la primera computadora electrónica en la Universidad Nacional Autónoma de México, en el Centro de Cómputo Electrónico, ahora Instituto de Investigaciones de Matemáticas Aplicadas y Sistemas (IIMAS), pero fue hasta 1970 cuando se iniciaron las actividades de **Inteligencia Artificial**, también en el Centro de Cómputo Electrónico, al organizarse una serie de conferencias de *Lisp*, en las que participaron Minsky y McCarthy, comenzando a despertar en los investigadores mexicanos el interés por esta nueva área de la computación. A partir de este hecho salieron a estudiar al extranjero los primeros especialistas mexicanos en **Inteligencia Artificial**.

En 1973 se incorpora por primera vez un curso de **Inteligencia Artificial** en la Facultad de Ciencias; en la UNAM se concentró en su inicio todo el trabajo de **Inteligencia Artificial**.

En 1983 se organizó la primera Reunión Nacional de **Inteligencia Artificial** en la ciudad de Jalapa, Ver., con representantes de la UNAM, IPN, ITESM, UDLA y UAM-I; para mediados de los 80's existía ya un gran interés en el campo formándose nuevos grupos de investigación como el de la Fundación Arturo Rosenbleuth, el ITESM, la UDLA y UAM-I, los que junto con la UNAM formaron la Sociedad Mexicana de **Inteligencia Artificial**, que dio mayor solidez y difusión a los trabajos desarrollados en el país.

En 1986 se hacen los primeros desarrollos para el sector productivo, uno es el sistema BYTEC y otro un prototipo para Bancomer, iniciando la comercialización de las aplicaciones de **Inteligencia Artificial**.

En 1987 se presenta en la Facultad de Contaduría y Administración la primera investigación referente a **Inteligencia Artificial**.

5. Objetivos de la Inteligencia Artificial.

Como su propia definición lo indica su objetivo principal es reproducir lo que se entiende por razonamiento y acciones inteligentes del humano con recursos artificiales, haciendo inteligentes los procedimientos que realizan, para de esta forma comprender los principios que hacen posible la inteligencia humana y dar también una nueva y mayor utilidad a las computadoras en beneficio del hombre mismo, de lo que Rauch afirma: " los **Sistemas Inteligentes** no sustituyen a las personas, más bien, aumentan su potencial". [Rauch 89].

Con el desarrollo de **Sistemas Inteligentes** se logra dar solución a problemas que requerirían una cantidad significativa de tiempo e intervención del hombre.

6. Disciplinas relacionadas con la Inteligencia Artificial.

Para alcanzar los objetivos anteriores, la **Inteligencia Artificial** actúa conjuntamente con otras ciencias o disciplinas, sus principales relaciones son con la psicología, la filosofía, la lingüística, la lógica y la ingeniería eléctrica.

Con la psicología se relaciona en cuanto a la comprensión del pensamiento humano, los investigadores hacen hipótesis sobre el proceso de solución de problemas y el comportamiento humano, que se representan en modelos computacionales a través de las técnicas y desarrollos de **Inteligencia Artificial**, verificando el razonamiento.

La filosofía apoya con sus estudios de las relaciones existentes entre el pensamiento y el comportamiento humano tanto físico como mental, interrelacionando el aspecto mental y físico de los individuos.

La lingüística aporta datos para el procesamiento del *lenguaje natural* tanto escrito como hablado.

La lógica contribuye en la *representación del conocimiento.*

La ingeniería eléctrica contribuye en la elaboración física de mecanismos y dispositivos para implementar los diseños de **Inteligencia Artificial.**

Con el apoyo de estas disciplinas la **Inteligencia Artificial** ha construido modelos de gran utilidad para otras ciencias como medicina, biología, química, matemáticas, ergonomía, etc..

7. Principales ramas de aplicación de la Inteligencia Artificial.

El desarrollo de la **Inteligencia Artificial** ha constituido nuevas vertientes en las áreas computacionales, las que están totalmente basadas en los principios de la **Inteligencia Artificial,** éstas son : Robótica, Sistemas Expertos, Comprensión del Lenguaje Natural, Visión por Computadora, Redes Neuronales, Algoritmos Genéticos y Lógica Difusa.

7.1 Robótica.

Esta área involucra programas que manipulan objetos por medio de los dispositivos de un *robot.* Su principal aplicación es industrial, en la cual se utilizan *robots* para la automatización de tareas repetitivas, la mayoría se limitan a realizar una secuencia idéntica de movimientos aprendidos, sin desarrollar nuevos conocimientos ni poseer otra inteligencia, pero actualmente se está trabajando en la nueva generación de *robots* que son capaces de percibir el ambiente y planificar su actividad, utilizan una cámara de TV para tener visión que manda la información visual al programa, que puede reconocer sombras, objetos y cambios de tonalidades. Es una de las áreas de **Inteligencia Artificial** más comercializada.

Los *robots* inteligentes combinan la destreza física de locomoción y manipulación que les caracteriza, con las habilidades perceptivas provenientes de los sensores y la capacidad de razonamiento, esas habilidades que pueden definirse como acción, percepción y razonamiento, le permiten interactuar con el medio ambiente.

A través de la acción el *robot* puede modificar su entorno mediante el desplazamiento de objetos o del propio *robot*.

La percepción retroalimenta al *robot* al darse cuenta de los efectos de sus desplazamientos para crear la descripción inicial del entorno y su situación con respecto al estado final esperado.

Los procesos de razonamiento le permiten elegir las acciones que debe de llevar a cabo para realizar la tarea indicada, hace una planificación para generar un plan de acción. Por medio del razonamiento puede hacer modificaciones al plan inicial si al ejecutarlo no se obtiene lo esperado.

Los mecanismos de acción con que cuenta un *robot* son de dos tipos: para desplazarse portando objetos o sólo para mover los objetos, lo que da origen a dos formas de desplazamiento en un *robot*, la locomoción que permite el desplazamiento completo del *robot* por ruedas o patas, y la manipulación que le permite manejar objetos por medio de brazos articulados. El potencial máximo del *robot* se alcanza con la combinación de las dos formas de desplazamiento.

Los sensores que utiliza el *robot* para obtener información acerca del medio que le rodea pueden ser ópticos, auditivos o de contacto con la visión utiliza imágenes para obtener la información sobre su espacio y los objetos, la visión en dos o tres dimensiones, tiene como ventaja el que no necesita tener contacto con los objetos por lo que no modifica su percepción primitiva. Los otros sensores le proporcionan la distancia que tiene con respecto al punto emisor.

Los *robots* se han hecho para realizar tareas muy especificas y limitadas (como cargar y descargar mercancia, ensamblar piezas, hacer aseo), la complejidad de un *robot* radica principalmente en hacerlo autónomo.

7.2 Sistemas Expertos (SE).

Dentro de la **Inteligencia Artificial** el trabajo hecho de *SE* es el más adelantado. Se trata de representar el comportamiento de un humano experto para la solución de problemas especificos, en el que no hay un *algoritmo* para realizarlo. Es requerida una gran *base de conocimientos* proporcionada por el experto humano, quien aporta sus conocimientos y experiencia. Los *SE* para su funcionamiento requieren de una arquitectura especifica, con una forma de incorporar *conocimiento* e interactuar con los usuarios. Actúa como intermediario entre el humano experto en la adquisición de sus *conocimientos* y el usuario que le consulta.

El *SE* debe aplicar los *conocimientos* de igual manera a como lo haria el experto humano, pudiendo resolver los problemas tan bien o mejor que el experto humano.

El experto humano es la persona que tiene conocimientos profundos de un tema especifico y cuenta con experiencia para resolver un problema en particular.

Los *SE* difieren de los programas de cómputo convencional por la manera de incorporar el *conocimiento* y en la forma de interactuar con los usuarios adoptando un comportamiento similar al del experto humano.

Los *SE* tienen la capacidad de razonar heuristicamente utilizando reglas de los expertos humanos, interactuar eficazmente y en lenguaje accesible con las personas, manipular expresiones simbólicas y razonar sobre éstas, funcionar con datos ambiguos y reglas

imprecisas, contemplar múltiples hipótesis alternativas, explicar las preguntas que realiza y dar justificación a las conclusiones obtenidas.

Los componentes principales de los *SE* son los mecanismos de *inferencia* y la *base de conocimientos.*

Los *SE* son utilizados con éxito en tareas de diagnóstico médico, químicos, de minerales.

Entre los *SE* desarrollados están:

DENDRAL, que se creó en 1969, en la Universidad de Stanford, para el estudio de estructuras moleculares en compuestos orgánicos y de resonancia magnética, se utiliza en la industria farmacéutica en Estados Unidos.

MYCIN, se desarrolló también en la U. de Stanford, en 1970, se aplica en el campo de la medicina. Se utiliza para el diagnóstico de infecciones bacteriales y para prescribir sus medicamentos, cuenta con alrededor de 450 *reglas de producción.*

PROSPECTOR, desarrollado en 1976, se utilizó en la minería y ayudó al descubrimiento de un yacimiento de molibdeno en Washington. El *SE* utiliza una combinación de *reglas de producción* y de *redes semánticas* para la *representación del conocimiento.*

CSG-2000, desarrollado en 1987, por la M.C.C. Marina Toriz Garcia, es un *sistema experto* hecho con el propósito de dar solución a la problemática económica nacional a través de política moderna.

7.3 Comprensión del lenguaje natural.

Una de las metas de la **Inteligencia Artificial** ha sido la creación de programas que sean capaces de entender y reproducir el lenguaje humano, tanto escrito como hablado. Se han

desarrollado técnicas para estructurar la *sintaxis* y *semántica* del lenguaje, pero existen además los aspectos de contexto, información sobre el idioma y otros difíciles de representar que hacen aún imposible lograr su entendimiento real para la máquina.

Hay programas que responden a preguntas simples así como traductores, pero que sólo reconocen un vocabulario muy restringido.

Los sistemas de *lenguaje natural* tratan de comprender el idioma en que se expresa el usuario, proporcionando un medio fácil de comunicación. Sirven de interfaz con las bases de datos. El lenguaje es una actividad simbólica y estos sistemas procesan estos *símbolos* y los relacionan con sus significados. [Rauch 89].

El procesamiento de *lenguaje natural* tanto escrito como hablado requiere *conocimientos* acerca de lingüística, gramática, fonética, acústica, lógica y psicología entre otros, por todo ésto es uno de los desafíos más grandes que tiene la Inteligencia Artificial.

Las primeras ideas surgidas acerca del tratamiento del *lenguaje natural* escrito, que datan de 1946, se referían a hacer una traducción automática del texto, pareciendo un diccionario electrónico que traducía palabra por palabra, después comienzan a interesarse por la *semántica*, se buscan palabras clave en la frase para analizarla, en los años 70's se considera tanto el aspecto semántico como el sintáctico, en este periodo se crea el sistema SHRDLU.

Con la creación de los *marcos de referencia* y los *guiones* surgen nuevas vías para el desarrollado de estas aplicaciones, apoyando la *pragmática*[3].

Las aplicaciones que pueden realizarse con el procesamiento del *lenguaje natural* son de traducción automática, realización de resúmenes de textos, indexación automática (para

[3] La *pragmática* permite dar al *conocimiento* un significado de acuerdo con su aplicación en un tiempo y espacio determinados.

búsqueda de palabras clave en un texto), y la generación automática de textos.

El procesamiento del lenguaje hablado por su parte, permite una comunicación directa entre el hombre y la máquina dándole las ventajas de poder ocupar sus manos y su vista en otras actividades, ser un usuario a distancia, facilitar su acceso a personas con problemas de ceguera o minusválidos u otros problemas físicos.

7.4 Visión por computadora.

Una de las formas de lograr que la computadora perciba el ambiente que le rodea es a través de la visión, para la cual se han desarrollado métodos capaces de reconocer formas visuales simples, sin poder todavía reconocer una escena o una imagen completa para lo que se requiere un conjunto de *conocimientos* y razonamiento del contexto. Esta área participa estrechamente con la *robótica*. Su objetivo principal es interpretar imágenes, no crearlas.

Los sistemas de percepción visual, audible y táctil, pueden interpretar escenas plásticas o inferir acerca de la calidad u orientación física de los objetos. Estos sistemas necesitan comprender tanto los conceptos simbólicos como los esquemas. [Rauch 89].

El objetivo del desarrollo de *visión por computadora* es interpretar una imagen o una escena y relacionar esta interpretación con una acción sobre el entorno. La visión por computadora utiliza otras técnicas como el tratamiento de imágenes o el reconocimiento de formas.

El tratamiento de imágenes hace una transformación de las mismas considerándolas como señales multidimensionales.

El reconocimiento de formas consiste en identificar objetos de acuerdo con medidas tomadas a estos objetos.

Las etapas fundamentales de visión son: la adquisición de la imagen o escena, el pre-tratamiento, la extracción de parámetros, la descripción simbólica y *semántica* y la interpretación final de la escena o imagen.

La adquisición de la imagen o escena se hace a través de sensores como son cámaras de video, láser, scanner, de acuerdo con el tipo de receptor se clasifica la imagen.

Las imágenes se representan por una matriz de *pixeles*, que dan a cada punto de la imagen el valor del nivel de gris asociado y el color.

Las imágenes utilizadas son generalmente binarias, donde cada *pixel* es negro o blanco, debido a que la potencia de cálculo y el espacio de memoria requeridos son por mucho menores que con imágenes multicolor.

El pre-tratamiento de la imagen junto con la extracción de parámetros dan información acerca de los contornos (cambios de intensidad luminosa) sirven para determinar el borde de los objetos, las texturas (muestran las variaciones locales de la intensidad del color), las distancias (para calcular la forma real del objeto en el espacio), el movimiento y las interacciones entre los objetos y el observador.

Durante la segmentación de la imagen en regiones, se considera que en cada región existe ciertas características homogéneas en cuanto al nivel de gris, la textura, la forma y el color, la segmentación puede ser utilizando los *pixeles* o los contornos.

Para la descripción *semántica* de la escena mediante niveles de abstracción se utilizan diversos tipos de *conocimiento* como modelos de objetos y de escenas apoyándose en el uso de ciencias físicas.

La última etapa del sistema es la interpretación de la imagen o escena, en la que se hacen corresponder los modelos prerregistrados de objetos, los que forman la *base de conocimientos*, con los parámetros extraídos de la nueva imagen.

La *visión por computadora* fue una de las primeras aplicaciones de la Inteligencia Artificial en la industria y continúa creciendo, sus aplicaciones principales son referentes a la inspección y guía automáticas.

7.5 Redes Neuronales.

Es una de las áreas de especialización de la Inteligencia Artificial que se basa en las redes neuronales biológicas, representadas en modelos matemáticos por medio de estructuras más sencillas.

Están integradas por una gran cantidad de procesos simples (unidades o neuronas), cada uno con la posibilidad de tener una pequeña localidad de memoria. Las unidades son unidas por canales de comunicación unidireccional, llamados conexiones, los cuales llevan datos numéricos. Las unidades operan únicamente sobre sus datos locales y sobre las entradas que reciben por sus conexiones.

Las *redes neuronales* son un recurso de procesamiento que está inspirado en el diseño y funcionamiento de la mente y sus componentes.

La mayoría de las *redes neuronales* tienen algún especie de regla de entrenamiento, por donde la ponderación de las conexiones es de acuerdo a los modelos presentados. Lo cual significa que las *redes neuronales* aprenden a través de ejemplos y exhiben alguna capacidad estructural por generalización.

Las neuronas se presentan frecuentemente como procesos elementales de señales no lineales, realizan discriminación, las *redes neuronales* tiene un alto grado de interconexión, la cual permite el paralelismo en gran medida. No tienen espacios de memoria desocupada que contengan datos o programas, a pesar de ello cada neurona es preprogramada y continuamente activada.

7.6 Algoritmos genéticos.

Son técnicas de búsqueda que se basan en la teoría de la evolución de Darwin, es decir, en los mecanismos de selección natural, según la cual los individuos más aptos son los que sobreviven, al adaptarse más fácilmente a los cambios que ocurren en su entorno.

La incorporación de esos mecanismos a la computación la hizo John Holland, con el objetivo de que las computadoras aprendieran por sí mismas, los llamó originalmente planes reproductivos y hasta 1975 les dio el nombre de *algoritmos genéticos*.

Los *algoritmos genéticos* se usan generalmente en problemas de optimización, los cuales cuentan con las características de: tener un espacio de búsqueda delimitado en un cierto rango y debe tener una función llamada de aptitud que pueda calificar qué tan buena o mala es una respuesta.

El *algoritmo genético* sólo se encarga de maximizar y tiene la posibilidad de castigar a las malas soluciones y premiar a las buenas, para que esas buenas se propaguen con rapidez.

Con relación a los mecanismos de la naturaleza, las diferentes alternativas de solución son comparadas con un conjunto de cromosomas, cada cromosoma será evaluado por la función de aptitud, para seleccionar a los mejores y proceder a la siguiente cruza o combinación y así sucesivamente hasta encontrar la solución final.

Esta es un área actualmente muy difundida dentro de la investigación en **Inteligencia Artificial** en el mundo.

7.7 Lógica difusa.

Es una tecnología para el desarrollo de sistemas de control que fue creada por Lotfi Zadeh en 1965. La lógica difusa está estrechamente relacionada con la lógica booleana, de la cual es derivada, pero que sostiene un concepto nuevo, el de las verdades parciales, es decir, habla de un valor que no es ni totalmente verdadero ni tampoco totalmente falso.

Su creador se basó en la situación de que la gente toma decisiones basándose en situaciones imprecisas y considerando información no numérica.

Para los procesos de control es necesario tomar decisiones contando con información precisa y considerando que las máquinas pueden tener una respuesta inmediata y mejor a las situaciones sería conveniente proporcionarles una forma similar al razonamiento humano espontáneo.

Introduce dentro de los sistemas los conceptos de palabras imprecisas, por lo que para referirse a una cantidad puede expresarse muchos, pocos, bastantes y en cuanto a medidas muy grande, pequeño, etc., sustituyendo a los valores numéricos.

Esta tecnología ha sido aceptada ampliamente en Japón, donde es muy utilizada para el control de procesos industriales a través de cámaras, en cambio en Estados Unidos ha tenido poca aceptación a pesar que fue el lugar en que se creó.

Las aplicaciones reales desarrolladas a través de las áreas de la Inteligencia Artificial anteriores son en los campos de:

Campo	Actividad
Industria	Control de calidad. Ensamblado. Selección de piezas y componentes. Monitorización. Control de procesos.
Finanzas	Auditoría. Gestión financiera. Diagnóstico de empresas. Evaluación de riesgos de inversión. Análisis de créditos y préstamos. Seguridad. Inspección.
Medicina	En la interpretación de imágenes. Ayuda en diagnósticos y tratamientos. Recuperación de enfermos. Visión y audición artificial. Monitorización de pacientes.
Militar y Aeronáutica	Planificación de tráfico aéreo. Ayuda al pilotaje. Identificación y seguimiento de objetivos. Evaluación de riesgos. Control de operaciones.
Recursos Naturales	Localización y recuperación de recursos naturales. Gestión de recursos naturales.

Cuadro 2. Aplicaciones de las áreas de la Inteligencia Artificial en diversos campos y /o actividades.

7.8 La quinta generación en la era computacional.

La quinta generación de computadoras es un proyecto que los japoneses emprendieron, desde los años 80's, apoyados por enormes inversiones financieras de su gobierno y empresas privadas.

El proyecto tiene como objetivo crear una máquina inteligente, que consiste en :
1) Interfaces inteligentes, incluyendo al *lenguaje natural* hablado y escrito, gráficos, etc., 2) uso de *bases de conocimientos* más que de datos y 3) utilización de las técnicas de *inferencia* y resolución de problemas heurísticos.

Las capacidades de las máquinas de quinta generación se basan en los desarrollos de **Inteligencia Artificial**, sustituyendo la arquitectura tradicional de Von Neumann, con el procesamiento paralelo logrando una velocidad de 1 billón de *inferencias* por segundo.

El lenguaje elegido para el proyecto es *Prolog*, para programar los métodos de *inferencia* necesarios.

A los lenguajes de **Inteligencia Artificial** se les ha dado el nombre de lenguajes de quinta generación.

El *hardware* de las computadoras de quinta generación se basa en los semiconductores de gran escala de integración (*VLSI*), los requerimientos de funcionamiento son expresados en número de inferencias lógicas por segundo (*lips*) de acuerdo al idioma inglés, el objetivo que se pretende alcanzar es de 50 a 100 millones de *lips* .

8. Herramientas para la Inteligencia Artificial.

Para lograr realizar los nuevos conceptos creados por la Inteligencia Artificial, los investigadores de los años 60's, tuvieron que generar sus propias herramientas de desarrollo, ya que con las que se contaba no era posible, porque sólo podían manipular información numérica y no simbólica como requería esta nueva área de la computación.

Lo primero que se desarrolló para dar solución a este problema fue IPL-1 por Newell, Shaw y Simon, de éste se derivó *Lisp*, creado por McCarthy, que era ya un lenguaje más formal que el anterior, después apareció *Prolog*, en los años 70's, por Alain Colmerauer en Marsella.

Han surgido diversos lenguajes especializados y de alto nivel sobre *Lisp* y *Prolog* para propósitos específicos como ayuda en la *representación de conocimientos* y la construcción de *sistemas expertos*.

8.1 Lisp.

Es el lenguaje de programación más utilizado en Inteligencia Artificial; se diseñó para que pudiera soportar la manipulación simbólica que no existía en los lenguajes comunes.

McCarthy lo creó pensando en desarrollar un lenguaje algebraico de procesamiento de listas para los trabajos de Inteligencia Artificial, a partir de eso surgieron los primeros dialectos de *Lisp* que se utilizaron en máquinas de IBM y DEC; entre 1960 y 1965 surgió *Lisp* 1.5, que fue la primera versión formal de *Lisp*; al inicio de los años 70's aparecen los dos dialectos principales de *Lisp*: MacLisp e InterLisp, el primero fue creado en el Massachusetts Institute of Technology (MIT) y el segundo en Cambridge.

MacLisp introdujo el concepto de funciones en *Lisp*, las que podían tener un número variable de argumentos, macros, arreglos, salidas no locales, velocidad aritmética, el primer compilador funcional de *Lisp* y la velocidad en la ejecución. Sus derivados son Franzlisp y Zetalisp.

Por su parte Interlisp introduce ideas en cuanto al ambiente y metodología de programación, aportó también la construcción de iteraciones, esta versión está mejor documentada y con más ventajas que Maclisp por lo que cuenta con un mayor número de usuarios y corre en máquinas DEC y Xerox.

Por la necesidad de estandarizar los dialectos existentes de *Lisp* surgió Common Lisp que pretende ser eficiente y portable a diferentes plataformas.

Para correr de manera eficiente los desarrollos en *Lisp*, se diseñó la máquina Lisp que se especializaba en los programas de *Lisp*, concepto que se desarrolló en 1960, Xerox sacó una serie de máquinas, que se denominó serie-D, utilizó las ideas aportadas por la máquina Lisp y las que se sacaron al mercado en 1981.

A finales de los años 70's el concepto de *programación orientada a objetos* comenzó a influir en la programación de **Inteligencia Artificial**, en especial en *Lisp*, donde se implementó el concepto de *herencia múltiple* y se crea Common Lisp Object System (CLOS).

8.2 Prolog.

Es un lenguaje de gran importancia en la **Inteligencia Artificial**, como su nombre lo dice es de orientación lógica[4] . Apareció varios años después que *Lisp*, en 1972 en Francia; es muy utilizado como herramienta dentro de los *sistemas expertos* y el procesamiento de *lenguaje natural*.

La programación de *Prolog* consiste de la declaración de *hechos*, la definición de reglas y preguntas y respuestas acerca de los *hechos* y sus relaciones. Utiliza la *sintaxis* de la *lógica de predicados* para la *representación de conocimientos*, con los que conforma su *base de conocimientos* y realiza las *inferencias* necesarias para poder responder a las preguntas hechas por el usuario. Su forma de *razonamiento* es *hacia atrás*, haciendo comparaciones para encontrar los *hechos* que cumplan con el objetivo o meta dado, por otro lado hace una *búsqueda en profundidad*, ya que va por una ruta o un nodo y si no es el correcto realiza un *backtracking* para regresar hasta el punto donde se estaba en lo correcto y tomar el siguiente nodo para evaluar.

Prolog no ha alcanzado aún la difusión que tiene *Lisp*, es utilizado principalmente en Europa y también en Japón para la programación de las máquinas de 5a. generación. Se hizo una versión la cual puede ser utilizada en cualquier tipo de computadora, incluyendo las microcomputadoras, es un lenguaje declarativo y es el representativo de los lenguajes de 5a. generación.

También se creó la versión de *Prolog* que implementa aspectos de *programación orientada a objetos*, se le llama OL(P).

[4] El nombre de Prolog corresponde a Programming in Logic o Programación Lógica.

8.3 Otras herramientas para Inteligencia Artificial.

8.3.1 Software.

A parte de estos dos lenguajes se han desarrollado otras herramientas, como las que muestra el cuadro 3, que no han tenido la repercusión de *Lisp* y *Prolog*, de los que han sido derivados, pero que han fomentado la mejora de los lenguajes de Inteligencia Artificial.

POP-2	Lenguaje de programación, utilizado en Inglaterra.
KEE	Herramienta desarrollada por IntelliCorp para desarrollos de Inteligencia Artificial.
LOOPS	Lenguaje de Programación desarrollado por Xerox para aplicaciones de Inteligencia Artificial.
OPS5	Lenguaje de programación utilizado para implementar *reglas de producción*, se basa en *Lisp*.
TRUCKING	Herramienta para la construcción de *SE* y también para el aprendizaje de Inteligencia Artificial.
LYGON	Lenguaje de programación lógica desarrollado en Australia en 1990.
OZ	Lenguaje para programación simbólica, mezcla características de *Prolog*, *Lisp* y Smalltalk.
PROMETHEUS	Lenguaje diseñado para aplicaciones de Inteligencia Artificial basado en programación lógica.
HELP-DEA	Herramienta para la programación de *robots* industriales.
ROSIE	Lenguaje de programación para desarrollar grandes *bases de conocimientos*, proporciona estructuras para *reglas de producción* y *marcos de referencia*.
UNITS	Lenguaje de representación de *conocimientos* para explorar *marcos de referencia*.

FRL	Lenguaje para la representación de información referente al *lenguaje natural*, se basa en *redes semánticas*.
OWL	Herramienta de representación de *conocimientos* que combina el *cálculo del predicado* con *marcos de referencia*, utilizado para comprensión de *lenguaje natural*.
DAWN	Lenguaje de *redes semánticas* empleado para desarrollos de *lenguaje natural* y *sistemas expertos*.
SAM	Lenguaje de representación de *marcos de referencia*.
KRL	Sistema para analizar *guiones*, desarrollado por Xerox.
CLP	Es como *Prolog* derivado de la programación lógica.

Cuadro 3. Herramientas para la Inteligencia Artificial.

8.3.2 Hardware.

El avance que se ha dado a nivel de *hardware* en la computación ha hecho que la **Inteligencia Artificial** ya no sea sólo un desarrollo de laboratorio, sino que pueda salir comercialmente o para aplicación práctica en una mayor cantidad de actividades.

Se han desarrollado componentes para sintetizadores de voz y para manejo de imágenes, las máquinas empleadas para **Inteligencia Artificial** necesitan tener procesadores de 32 bits y *arquitecturas RISC* para lograr funcionar eficazmente, así como pantallas gráficas de alta definición y cantidades grandes de memoria.

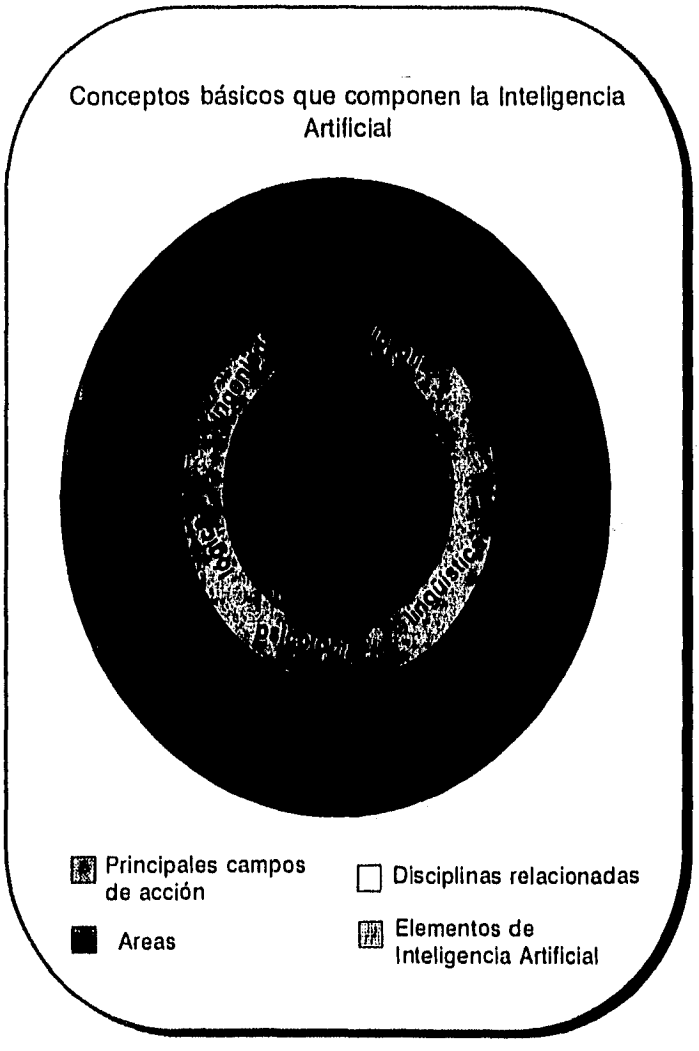


Figura 1. Esquema general de la Inteligencia Artificial.

9. Ingeniería de software para Sistemas Inteligentes.

El ciclo de vida de todos los sistemas está integrado por las siguientes etapas:

Análisis.
Diseño.
Programación.
Implantación.
Mantenimiento.

Dentro de los **Sistemas Inteligentes** cada una de esas etapas es cubierta de la siguiente manera:

1. Análisis de **Sistemas Inteligentes**.

En la cual se hace un estudio preliminar, en la que se identifica y define claramente el problema, las expectativas del sistema, las alternativas de solución, los recursos requeridos, se determina si la solución a través de un **Sistema Inteligente** es la más adecuada.

Después del estudio preliminar, si se determinó seguir con el desarrollo de un **Sistema Inteligente**, es definido el tipo de *conocimiento* que será almacenado y la forma de adquisición del mismo, es decir, se define la *representación del conocimiento* más apropiada.

También se determinan las expectativas que tiene el usuario en cuanto al sistema, se estudian las alternativas encontradas para lograr el comportamiento deseado del sistemas eligiendo la más adecuada de acuerdo a la situación especifica.

2. Diseño de **Sistemas Inteligentes.**

De acuerdo con los resultados obtenidos por la etapa anterior, en esta etapa se define el tipo de búsqueda inteligente que es más adecuado para operar con la *base de conocimientos*, se define la modularidad, la interfaz que resulte más conveniente para la interacción con el usuario.

3. Programación de los **Sistemas Inteligentes.**

En esta etapa se lleva a cabo la codificación del **Sistema Inteligente** en el lenguaje o con la herramienta seleccionada. El lenguaje C++ Inteligente ha sido pensado para poder proporcionar facilidades en esta etapa de desarrollo de los **Sistemas Inteligentes.**

4. Implantación de **Sistemas Inteligentes.**

Después de hacer las pruebas pertinentes así como las correcciones necesarias antes de ponerlo en operación, finalmente el sistema correcto tal como el usuario espera es puesto en marcha.

Las pruebas se hacen con personal diferente al que realizó el sistema, pueden intervenir los usuario finales u otras personas, con el objeto de verificar si se consideraron todos los aspectos inicialmente establecidos, si está a prueba de fallas, su facilidad de uso y en general encontrar todos los errores posibles.

5. Mantenimiento de **Sistemas Inteligentes**.

Es necesario que el sistema se adapte a las situaciones cambiantes del entorno en que opera, los **Sistemas Inteligentes** tienen como una de sus características principales el poderse adaptar de manera fácil a este tipo de cambios, es una etapa que continuará durante toda la vida útil del sistema, ayuda a mejoramientos subsecuentes. Las *bases de conocimientos* pueden ser expandidas, pueden modificarse las interfaces, o realizar cualquier otro cambio en el *hardware* o *software* del sistema.

Para esta etapa es una gran ventaja contar con una documentación lo más amplia y detallada posible del análisis, diseño y programación del sistema.

Capítulo II.

El conocimiento como parte del análisis de Sistemas Inteligentes.

Capítulo II. El conocimiento como parte del análisis de Sistemas Inteligentes.

1. Introducción.

Entre las decisiones principales que se enfrentan en el desarrollo de **Sistemas Inteligentes** es la referente a la elección de la mejor forma de *representación del conocimiento*, para cada problema en particular se debe buscar la representación más adecuada, de esta elección depende la forma en que se explotará el *conocimiento* existente hacia la solución correcta del problema. Cada *representación del conocimiento* va relacionada estrechamente con su explotación. La elección de la *representación del conocimiento* forma parte de la etapa de análisis dentro de la ingeniería de *software* para **Sistemas Inteligentes**

Los *hechos* y demás *conocimientos* contenidos en los modelos desarrollados para su representación deben poder ser utilizados en razonamientos, lo que significa que la *base de conocimientos* pueda ser empleada por los métodos de *inferencia*, basados en búsquedas.

Cualquier forma de *representación del conocimiento* debe cumplir con la función de poder capturar las características esenciales del problema y hacer accesible la información para el proceso de su solución.

El *conocimiento* humano ha sido estudiado por psicólogos, filósofos y educadores, entre otros, y ahora también el campo de la computación se involucra en estos estudios a través de la **Inteligencia Artificial** con el fin de conocer los procesos que se llevan a cabo en el interior de la mente durante la adquisición, almacenamiento, recuperación y utilización del *conocimiento*.

La Inteligencia Artificial ve al *conocimiento* como la capacidad del ser humano de entender, aprender, percibir, descubrir, inferir, obtener experiencia y organizar la información adquirida para dar solución a un problema.

2. El conocimiento humano.

Existen cuatro tipos de *conocimiento* [Feigenbaum 89] :

1) De objetos. Se refiere al *conocimiento* de los objetos del mundo real que encontramos a nuestro alrededor, se representan por objetos, clases o categorías de objetos y sus descripciones, p.e. una factura, un cheque.

2) De eventos. Son las acciones o acontecimientos del mundo real, pera su representación se debe indicar la secuencia en que ocurren y su relación de causa y efecto, p.e. la contratación de personal en el departamento de producción, la capacitación del personal.

3) Del funcionamiento. Es el *conocimiento* que involucra más que el mismo *conocimiento* de los objetos y eventos, incluyendo también el cómo utilizarlos, cómo hacer las cosas p.e. el establecimiento de políticas y normas en una organización.

4) Metaconocimiento. Se refiere a la confianza que se tiene en el propio *conocimiento*, el *conocimiento* que tenemos de nuestro propio *conocimiento*, de nuestros procesos del pensamiento, experiencia y sabiduría.

El límite entre estas formas de *conocimiento*, así como sus variaciones, son definidos por la psicología, la Inteligencia Artificial sólo se ocupa de su representación, para poder mostrar el comportamiento ocurrido en cada uno de estos *conocimientos*.

2.1 Aplicación del conocimiento humano dentro de los Sistemas Inteligentes.

La Inteligencia Artificial transcribe ese *conocimiento* humano en forma de estructuras que pueden ser exploradas por un sistema computarizado de razonamiento en búsqueda de una solución, a lo que llamamos *representación del conocimiento.*

La importancia de la *representación del conocimiento* radica en que de ésta se desprende la utilización del propio *conocimiento*, la cual está compuesta por tres operaciones básicas : [Feigenbaum 89].

1) Adquisición del *conocimiento.*
2) Recuperación del *conocimiento.*
3) Razonamiento.

1) Adquisición del *conocimiento.* El aprendizaje, que corresponde a la acumulación de *conocimientos* es uno de los objetivos de los sistemas de Inteligencia Artificial, esta actividad involucra otras como la adición de nuevos *hechos* a la *base de conocimientos* y la relación de estas estructuras nuevas con las ya existentes para su interacción, todo eso para lograr su reutilización y recuperación cuando sea necesario.

La adquisición del *conocimiento* puede hacerse de diversas fuentes como: de los humanos expertos, de los libros o los documentos en general, de los sensores y de los archivos computacionales, se extrae el *conocimiento* de estas fuentes y se organiza en estructuras de *representación del conocimiento.*

2) Recuperación del *conocimiento.* Para llevar a cabo esta fase es necesario determinar qué *conocimiento* es relevante para la solución de un problema, pero dentro de un sistema de cómputo esto es aún muy difícil, debido a que en el humano no se conoce con certeza este mecanismo, por lo que se ha hecho sólo en base a ideas que se tienen acerca del

funcionamiento de la memoria humana. Concluyendo para su realización en una serie de relaciones, ligaciones y agrupaciones entre las *estructuras de datos simbólicos* relacionados.

3) Razonamiento. Cuando el sistema requiere la realización de una tarea que no tiene perfectamente definida como hacer, debe utilizar el razonamiento y verificar lo que necesita conocer o utilizar de sus *conocimientos* previos. El sistema debe ser capaz de deducir y comprobar diversos *hechos* nuevos más allá de los que tiene.

El razonamiento puede hacerse en alguna de las cinco formas siguientes : [Feigenbaum 89].

Razonamiento formal. Indica la manipulación de las estructuras de datos para deducir otras nuevas mediante reglas de *inferencia*. La lógica matemática es utilizada como representación para este tipo de razonamiento.

Razonamiento procedural. Utiliza simulación de acuerdo con un modelo para la solución de problemas o respuesta a preguntas.

Razonamiento por analogía. Esta forma hace que los sistemas de Inteligencia Artificial lleguen a un resultado a través de establecer relaciones de semejanza o analogía entre sus *conocimientos* y el problema o pregunta a resolver.

Razonamiento por generalización y abstracción. Esta forma de razonamiento al igual que la anterior son utilizadas por la mente humana directamente, por lo que se denominan como procesos de razonamiento natural en los humanos, este último razonamiento no ha sido implementado aún con éxito en los programas de Inteligencia Artificial.

Metanivel de razonamiento. Esta forma de razonamiento utiliza el *conocimiento* acerca de lo que se conoce, sobre su extensión e importancia.

Los tres usos del *conocimiento* están interrelacionados entre si, ya que al adquirir un nuevo *conocimiento* el sistema debe saber como recuperarlo y utilizarlo más tarde para su razonamiento. El nuevo *conocimiento* adquirido se utilizará de la misma forma, creándose un ciclo de adquisición-recuperación-razonamiento.

3. Representaciones procedurales, declarativas y mixtas del conocimiento.

Los modelos para representar el *conocimiento* se clasifican en procedurales, declarativas y mixtas (Figura 2).

Las representaciones procedurales son aquellas que representan el *conocimiento* formado por los *hechos* acompañados de la indicación de cómo se utilizará, muestran cómo hacer las cosas; establece una relación entre la información y su control. Cuando se procesa el *conocimiento* el control de cada proceso determina qué información debe tomar para evaluar y utilizar posteriormente.

Los *autómatas finitos* y programas de cómputo son esquemas de este tipo.

En estas representaciones se muestran los métodos o procedimientos que se aplicarán al *conocimiento*.

Entre sus ventajas se encuentra que :

- La *representación del conocimiento* y cómo hacer las cosas se facilita.
- Se utiliza para representar el *conocimiento* que en los esquemas declarativos resultaría difícil.
- Pueden representarse de manera eficiente los métodos heurísticos.
- Es fácil de entender y codificar.

Se habla de representaciones declarativas cuando se trata de una descripción del *conocimiento* respondiendo a la pregunta ¿qué se representará?, se representa la información o *conocimiento* manejado con estrategias de propósito general que logran un control de alto nivel en su manejo, lo que permite ser utilizado por diversos métodos.

Estas formas proporcionan flexibilidad y economía en el uso de estructuras, al igual que en la obtención de la totalidad de sus deducciones y certeza en las mismas.

Sus principales ventajas son :

- Los *hechos* sólo se almacenan una vez, sin tomar en cuenta las diferentes formas de uso que se les puede dar.
- Se pueden añadir nuevos *hechos* con facilidad, sin tener que cambiar los hechos anteriores ni los procedimientos que los utilizan.

Actualmente a pesar de los atributos que cada una puede tener de manera individual (Cuadro 4), se tiende a utilizar representaciones mixtas, es decir, que incluyen características de una y otra, asociando en la misma estructura la especificación del *conocimiento* y los mecanismos de su utilización.

Dentro de las formas mixtas se encuentran los *marcos de referencia (frames)* y los *guiones (scripts)*.

Figura 2. Representaciones del conocimiento.

Todas las *representaciones del conocimiento* deben tener como características principales:

Modularidad.

Esta característica hace que el *conocimiento* se represente dentro de un programa como partes de estructuras de datos, lo que permite añadir, modificar o eliminar estructuras individuales al sistema con la mayor independencia posible y hacer que las consecuencias que ésto produce sean lo más transparentes posible en el manejo del *conocimiento*.

El no representar modularmente el *conocimiento* traería como consecuencia una interdependencia entre todos los elementos, procesos y *hechos* representados, existiendo un solo control en el que se basaría todo el *conocimiento* representado, sin tener cada *hecho* su propio significado, el cual se especifica al almacenarse, eliminarse o modificarse de manera independiente del resto del esquema.

La modularidad también proporciona mayor facilidad al desarrollador para entender la *representación del conocimiento*, repercutiendo en el diseño, implementación, adquisición del *conocimiento*, funcionamiento e interacción de procesos.

Conocimiento explícito.
Los esquemas de *representación del conocimiento* deben mostrar qué *conocimiento* ha sido proporcionado directamente al sistema, cuál se ha construido, y cuál se accesa directamente por el programador y el sistema.

El hecho de que el *conocimiento* sea explícito permite hacer una representación global de los *hechos* almacenados y su uso para múltiples propósitos.

En el caso contrario, es decir, con un *conocimiento* implícito, el esquema de forma oculta trabaja con el *conocimiento*, donde el sistema no muestra la manera cómo lo maneja y sin permitir su uso en otras circunstancias.

Expresión y eficiencia.
Los modelos de representación permiten expresar con sus elementos todo o la gran mayoría del *conocimiento* que el usuario necesita para dar solución a los problemas planteados.

Debe evitarse la limitación al representar o capturar las características esenciales del *conocimiento* que solucionará el problema, para así contar con la mayor cantidad posible de referencias acerca del problema.

Es una de las características más importantes de los esquemas de *representación del conocimiento*, por lo que se debe buscar siempre el más adecuado a los requerimientos de cada problema a solucionar. Elegir la mejor forma para representar todos los aspectos que pueden ayudar en la solución del problema no siempre resulta ser eficiente, por lo que debe buscarse un equilibrio siempre entre la representación idónea y la más adecuada.

Declarativas	Procedurales
Hace énfasis en el conocimiento específico de un dominio.	Se ocupa en la forma en que se dará solución.
Sus técnicas son aplicables a las situaciones del dominio.	Con técnicas de soluciones generales.
Es amigable con el usuario y fácil de entender.	Se orienta hacia la eficiencia en la solución, facilita la representación del control del conocimiento.
Paralelismo natural, pero en contreposición por búsquedas innecesarias.	Paralelismo restringido y frecuentemente especificado por el usuario.
Su control es transparente para el usuario.	El control es especificado por el usuario.

Cuadro 4. Atributos de las representaciones declarativas y procedurales del conocimiento. [Wah 90].

4. Modelos de representación del conocimiento.

4.1 Representaciones lógicas.

Estas formas de representaciones fueron las primeras en utilizarse en la **Inteligencia Artificial** para representar el *conocimiento*.

Es una forma muy antigua de representar *hechos*, tiene sus orígenes en Grecia, donde comenzaron a utilizarla desde un punto de vista filosófico, para realizar el análisis del comportamiento racional humano. Después estos estudios fueron retomados y formalizados por Boole, Frege y Russell quienes les dieron un enfoque matemático, en la **Inteligencia Artificial**, McCarthy introdujo esta forma de representación.

El esquema general de un proceso de razonamiento lógico es como se indica en la figura 3:

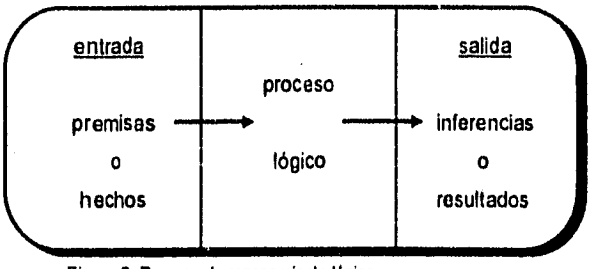


Figura 3. Proceso de razonamiento lógico.

Su uso en la **Inteligencia Artificial** se debe a que es una forma confiable para obtener *conocimiento* nuevo a partir del ya existente, por medio de la *deducción* matemática, se pueden derivar mecánicamente hechos nuevos de otros anteriores usando versiones automatizadas de técnicas provistas de teoremas o programas hechos para determinar automáticamente la validación de una nueva posición.

Estas formas de *representación lógica* operan mejor cuando el número de *hechos* o *conocimientos* con que trabaja es reducido, cuando aumentan encuentra dificultades porque existe un número mayor de *hechos* y combinaciones por evaluar.

Las formas más importantes de la *representación lógica* son : la *lógica de proposiciones* y la *lógica de predicados* que es la más aplicada en la Inteligencia Artificial.

Estas formas evalúan las proposiciones para determinar su validez de acuerdo con su *sintaxis*, hacen una manipulación sintáctica con sus fórmulas lógicas para poder aplicar la *deducción*, representada por las reglas de *inferencia*. Se busca siempre obtener un resultado válido.

Algunos autores se refieren a estas dos formas de *representación del conocimiento* como cálculo de proposiciones y del predicado, dando a la palabra cálculo el sentido computacional, y no del álgebra matemática[5] .

4.1.1 Lógica proposicional.

La proposición es un enunciado que se evalúa de acuerdo con su *sintaxis* para obtener sólo dos posibles valores: verdadero o falso, para llegar a formar después una *premisa* que derivará nuevas proposiciones o *inferencias*, que se someterán a reglas para determinar la verdad o falsedad de una nueva proposición.

La proposición está integrada por variables proposicionales y las relaciones que existen entre éstas.

[5] En computación la palabra calculable se refiere a aquello que puede ser manipulado por el sistema de cómputo.

Existen proposiciones simples y compuestas, las simples indican una sola idea y las compuestas se forman a partir de la unión de dos o más simples por medio de conectores.

Los conectores utilizados en la lógica son :

Conector	Símbolo
And	&, ∩
Or	∪
Not	¬,~
Implica	⇒,→,⊃,if _ then
Equivalente	≡,⇔,↔,if _ and only _ if

Figura 4. Conectores lógicos.

Para la evaluación de las proposiciones, tanto simples como compuestas, se utilizan procedimientos basados en *axiomas* y en las reglas de transformación, los que dan como resultado un sistema de *inferencia*.

La tabla de verdad (Figura 5), se forma en base a la evaluación hecha a una proposición simple o compuesta que dan como resultado un valor verdadero o falso.

A	B	A ∩ B	A ∪ B	A ¬ B	A ⇒ B	A ⇔ B
V	V	V	V	F	V	V
V	F	F	V	F	F	F
F	V	F	V	V	V	F
F	F	F	F	V	V	V

Figura 5. Tabla de verdad para el uso de los conectores lógicos.

DONDE :

$A \cap B$	Resulta verdad (V) sólo si A y B son verdaderas, sino es falso (F).
$A \cup B$	Resulta verdad si A o B son verdaderas o si lo son ambas, sino es falso.
$A \neg B$	Resulta verdad si A es falsa , si es verdadera resulta falso.
$A \Rightarrow B$	Resulta verdad si B es verdadera o si A es falsa.
$A \Leftrightarrow B$	Resulta verdad sólo si A y B son verdaderas o si ambas son falsas, si son distintas es falso.

El sistema o mecanismo de *inferencia* utilizados en la *lógica de proposiciones* se compone de las fórmulas :
Modus ponens.
Es una *tautología* para la inferencia la cual sigue la siguiente forma:
Si, si P entonces Q y P, entonces Q.

Modus tollens.
Esta *tautología* funciona de manera similar que la anterior, pero en los casos de negación:
Si, si P entonces Q y no P entonces no Q.

Al combinar sintácticamente las variables o proposiciones simples y los conectores se obtienen las proposiciones compuestas o enunciados de *lógica proposicional* que son iguales a las expresiones matemáticas utilizadas en el *álgebra booleana.*

Las proposiciones por separado no son relevantes para la Inteligencia Artificial, ya que su interés radica en la interrelación que puede darse con otras proposiciones, que es en lo que se basan los problemas del mundo real.

Cuando es verdadera una proposición para todos los posibles valores que pueda tener en sus variables se dice que es una *tautología*; y es una falacia o contradicción, si resulta falsa para todos sus valores asignados.

En la lógica proposicional se encuentran las primeras reglas de *inferencia* que permiten la *deducción* de nuevos *conocimientos* partiendo de los *conocimientos* previamente almacenados.

4.1.2 Cálculo del predicado.

La *lógica de proposiciones* sólo permite una representación muy limitada del *conocimiento* para la Inteligencia Artificial, por lo que se desarrolló el *cálculo del predicado*, que está basado en la *lógica de proposiciones*, utiliza sus conceptos y reglas, pero que permite dar un mayor detalle al *conocimiento*.

Mientras que los elementos por separado de una proposición no significan nada para la Inteligencia Artificial, en el *cálculo del predicado* sus partes componentes si son relevantes, pueden especificarse objetos y sus relaciones así como generalizar estas relaciones sobre clases de objetos.

Los elementos manejados dentro del *cálculo del predicado* en una proposición son básicamente :

Argumentos o términos. Son las variables o constantes a evaluar, las que representan objetos, entidades o individuos.

Predicado. Indica la propiedad del argumento o la relación entre varios términos.

p.e. área-funcional (control de calidad), puesto (Sr. Ruíz, auditor).
Donde los términos son control de calidad en uno y Sr. Ruíz y auditor en el otro y los *predicados* son área-funcional y puesto.

Es posible combinar varios *predicados* a través de funciones que permiten representar transformaciones, en una función el resultado es un elemento más, un objeto, y en un *predicado* el resultado es del tipo verdadero o falso. Los argumentos de una función son variables, constantes u otras funciones. Es decir, en una función el resultado es otro elemento que será utilizado por un *predicado* o por otra función, en cambio en un *predicado* el resultado obtenido, es un resultado final, una respuesta de verdadero o falso, no un elemento más.

p.e. comisión (ventas(empleado1, enero),25000, 15%).
Aquí se combinan dos *predicados*: comisión y ventas, sin embargo, ventas se convierte en una función del *predicado* comisión, ya que le regresa un valor o elemento más, el monto de las ventas, que operará el *predicado* comisión.
Se lee: Si las ventas del empleado1 en el mes de enero llegan a 25000 obtendrá una comisión del 15%.

El *cálculo del predicado* permite también el uso de los conectores de la *lógica de proposiciones*, y se introduce un nuevo concepto, los cuantificadores, que pueden ser existencial y universal.

p.e. ventas (empleado1, 25000) ∩ antigüedad (empleado1,12)-> premio (1200);
publicidad (productoA,1990) ∪ demanda (productoA,5000) ->abrir-mercado(zonab).

En estos dos ejemplos se utilizan los conectores lógicos AND (∩) y OR (∪) para unir *predicados*.

Cuantificadores:

∀ - para todo , cuantificador universal.

∃ - existe, cuantificador existencial.

p.e. ∀ empleado(y)->prestaciones(n, vales).
Aquí se utiliza el cuantificador universal para indicar que todos los empleados de la organización debe recibir ciertas prestaciones.

4.1.3 Lógica de primer orden.

Con la introducción de funciones a la *lógica de predicados* se crea la *lógica de primer orden.*

Como ya se mencionó, las funciones tienen argumentos de cuya evaluación no se obtienen los valores de verdadero o falso sino objetos individuales, cada argumento de la función puede tomarse como una variable, una constante o una función con sus propios argumentos.

En la *lógica de primer orden* se puede hacer cuantificación de objetos, variables, pero no de funciones o *predicados*, se utilizan los cuantificadores universal y existencial, además de que se pueden realizar también igualaciones.

Los tipos de representación lógica del *conocimiento* son útiles para la solución de problemas que requieren un tratamiento deductivo, pero la mayor parte de los problemas reales que la **Inteligencia Artificial** pretende solucionar son de naturaleza inductiva, por lo que es necesario apoyarse en otros esquemas.

Debido a que sólo representa el *conocimiento* sin mostrar el cómo se utilizará es una forma de *representación declarativa del conocimiento.*

Entre las ventajas de estas *representaciones lógicas* se enuncia que :

- Permite expresar el *conocimiento* de una manera que nosotros podamos entender muy bien.
- Existen métodos para determinar el significado de la expresión lo que lleva a un alto grado de precisión.
- Permite representar un *hecho* o el *conocimiento*, sin tomar en cuenta el uso del que será objeto.
- Es completamente modular, pudiendo añadir, modificar o eliminar *predicados* sin alterar totalmente la estructura.

4.2 Redes semánticas.

Esta forma de *representación del conocimiento* fue también una de las primeras en utilizarse, en 1968 la creó Ross Quillan. En un principio se utilizaron las redes para los modelos psicológicos de memoria asociativa, por lo que se le llamó memoria semántica, así como para el entendimiento del *lenguaje natural*.

Actualmente existen varias versiones de *redes semánticas*, pero en general, todas se basan en los mismos aspectos de esta representación.

Las *bases de conocimientos* para establecer las relaciones contienen un sistema de *estructuras simbólicas de datos* a las cuales se incorporan significados, por lo que se denominan semánticos.

En las *redes semánticas* se considera al *conocimiento* como un conjunto de asociaciones entre conceptos. Los conceptos son representados por *nodos* y sus asociaciones o relaciones por arcos o ligas que unen dichos *nodos*.

Puede definirse a una *red semántica* como un *grafo* con *nodos* y arcos, en el que los *nodos* representan conceptos y las ligas o arcos las relaciones de naturaleza *semántica* establecida entre los conceptos, lo que le da el sentido a la *red semántica* (Figura 6).

En general, los conceptos o *nodos* se refieren a cualquier sustantivo y las relaciones o arcos a verbos. Los sustantivos pueden ser individuales, referentes a una entidad (objeto, individuo), a un atributo, un estado o un evento o una clase de ellos.

Para las relaciones pueden utilizarse todos los verbos e incluso la pertenencia a un conjunto (es un), con lo que se pueden realizar todas las operaciones de teoría de conjuntos dentro de las *redes semánticas*, también existe el indicador de pertenencia de un atributo (tiene un).

Es posible considerar el modelo de *redes semánticas* como una teoría computacional para el entendimiento verbal del hombre.

En la representación gráfica de las *redes semánticas* los nodos se ilustran como puntos, cajas o círculos y los arcos como flechas.

Una de las características importantes de las *redes semánticas* es que permiten utilizar la *herencia*, de manera que los atributos o circunstancias de los *nodos* pueden heredarse a otros.

Las *redes semánticas* pueden hacerse con un nivel de detalle muy alto, depende del tipo de problema que se vaya a resolver a través de la misma. Cuando se trata de un problema general se requiere menos detalle que si fuese muy específico.

La variedad que existe de *redes semánticas* se debe a que para su representación y solución no existe una *semántica* formal como la de lógica, por lo que se crean diversos procedimientos con los que se realizan *inferencias* para dar solución al problema.

Los dos procedimientos más representativos son :
El de Quillan, llamado actuación extendida (Spreading actuation), éste consiste en un conjunto de procedimientos para manipular la red y hacer *inferencias* por pares de conceptos encontrando sus conexiones entre los *nodos* involucrados.

Comienza tomando dos *nodos*, que activan a los demás *nodos* que están interrelacionados o conectados con ellos, de manera que queda activada una zona alrededor del par de *nodos* originales, se busca una conexión entre las dos direcciones producidas, cuando simultáneamente se activa un *nodo* por estas dos direcciones, el programa debe de describir la ruta resultante.

El otro método de solución, utilizado por la mayoría de las *redes semánticas* es el de fragmento de red (fragment network), basado en parejas de estructuras de redes. Se construyen los fragmentos al representar una búsqueda o una consulta de un objeto aparejada a la red para saber si existe o no el objeto. Los *nodos* variables en el fragmento son unidos en el proceso al valor que deben tener para lograr un par perfecto.

En este método se pueden hacer *inferencias* durante el proceso de búsqueda creando una estructura de red que no está directamente expresada en la *red semántica* de origen.

La representación gráfica de las *redes semánticas* es sólo para que el desarrollador entienda mejor el proceso, ya que en la computación es necesario adaptarlo a los lenguajes existentes, por lo que muchas veces se utilizan sólo como representación visual de las relaciones y se combinan con otros métodos de representación.

El uso dado principalmente a esta forma de representación, es para los sistemas de comprensión del *lenguaje natural* y de diagnóstico médico, donde se hace una relación entre los síntomas presentados, sus causas y efectos, también se han desarrollado para la evaluación de minerales en la tierra.

Es una forma declarativa de *representación del conocimiento* porque no hace mención de los métodos que utilizan el *conocimiento* expresado.

Las ventajas que ofrece este modelo de representación son :

- Es de gran flexibilidad para incorporar nuevos *nodos* y sus relaciones.
- La representación gráfica visual que proporciona es sencilla de entender.
- El mecanismo de *herencia* que ofrece permite un mejor aprovechamiento de sus elementos y facilita el razonamiento.

Sus limitaciones principales son :

- No cuenta con una terminología estándar, sobre todo en los tipos de relaciones que pueden establecerse entre los *nodos*.
- Cuando la red es muy grande o ha crecido mucho su procesamiento resulta muy complicado, por el número de relaciones que deben evaluarse.
- No pueden representarse aspectos como el tiempo y la secuencia.

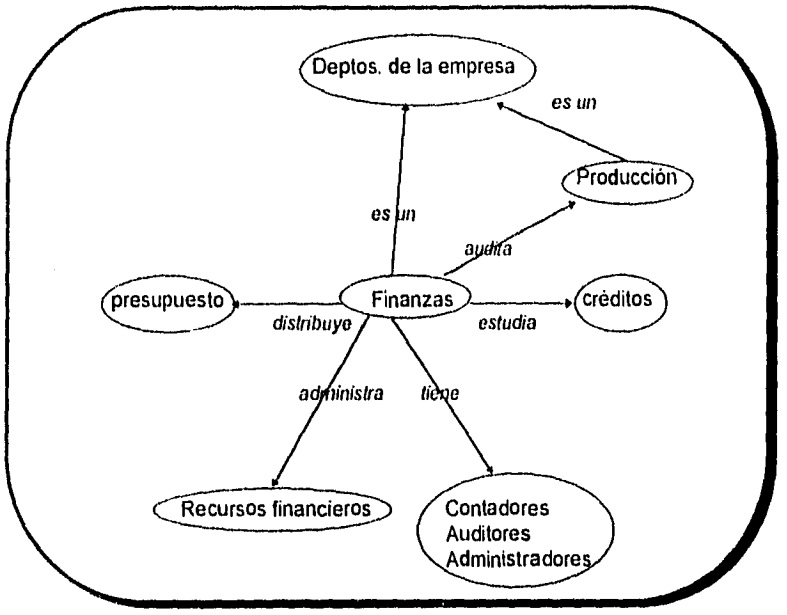


Figura 6. Red semántica del departamento de finanzas de una empresa.

4.3 Reglas de Producción.

Esta forma de *representación del conocimiento* fue creada en 1943 por L.. Post, para la representación del modelo cognoscitivo humano que se relaciona a las *reglas de producción* por la activación de las neuronas.

Se basa en la idea de representar el *conocimiento* como un conjunto de reglas con la forma de un par de condición - acción :

If condición **Then** acción.

p.e. Regla (R). **If** el ingreso de un trabajador es menor de 3 salarios mínimos **Then** no paga ISR.

Si la condición se cumple entonces se ejecuta u ocurre una acción o conclusión.

Cada *regla de producción* es una parte pequeña de *conocimiento* que debe cumplir con la forma antes descrita.

La formación de *reglas* del tipo situación/acción o síntoma/causa es adecuada para expresar los *conocimientos* que los especialistas de una área obtienen con la experiencia - reglas empíricas u ojo clínico-. Las *reglas* proporcionan un método conveniente de representar los *conocimientos* de los especialistas humanos ya que éstos utilizan muchas reglas aprendidas en la experiencia diaria que son aprovechadas y utilizadas en los *SE*.

Las *reglas de producción* expresan el *conocimiento* de una manera declarativa, puesto que es independiente de su uso. Las reglas declarativas que se refieren al *conocimiento*, integran todos los *hechos* y relaciones establecidas.

Las *reglas* pueden escribirse de una manera jerárquica, en donde las de mayor nivel son las que actúan sobre otras *reglas* y las de menor nivel son *reglas* que actúan sobre la *base de conocimientos*. A esas *reglas* que se aplican sobre otras *reglas* y no sobre objetos o hechos se les llama *metarreglas*, que son además *reglas* de tipo procedural.

Las *reglas de producción* en conjunto forman parte de un *sistema de producción*, integrado por cuatro partes :

1) Una base de *reglas* que corresponden al conjunto formado por las *reglas de producción*.

2) Una o más bases de *hechos*, sobre las que se ejecutan las condiciones y acciones de las *reglas*.

3) Un espacio en memoria que contiene los aspectos de relevancia de las *reglas de producción.*

Las *reglas de producción* están contenidas en un espacio de memoria denominado memoria de producción, además existe otro tipo de memoria, la memoria de trabajo, en la que se encuentran los objetivos a los que se desea llegar así como las *reglas de producción* que se van aplicando.

4) Un intérprete que lleva el control del sistema, determina qué regla de producción o tarea se ejecutará en seguida.

Una vez que el *conocimiento* se ha expresado en formato de *regla* un intérprete de *reglas* aplica las *reglas* adecuadas a determinadas situaciones, anota los resultados, y vuelve a aplicar las *reglas* en base a la nueva situación.

Cuando se ejecuta un programa, el intérprete de *reglas* lo estructura dinámicamente y aplica las *reglas* usando un procedimiento de adaptación de modelos que verifica si las condiciones especificadas en la parte IF de una *regla* (que visualiza las condiciones como modelos) se adaptan o coinciden con los modelos similares existentes en la *base de conocimientos* ubicada en alguna localidad de memoria de la computadora. Es decir, que las *reglas* se aplican según los principios de la coincidencia de modelos, permitiendo añadir, modificar o eliminar *reglas* con facilidad, sin preocuparse de su secuencia y consistencia, pudiendo también explorar y realizar rápidamente prototipos de sistemas complejos, débilmente especificados y sujetos a múltiples modificaciones. [Rauch 89].

En algunos *sistemas de producción* a las *reglas* se les asigna un coeficiente de verosimilitud, para su evaluación.

Los *sistemas de producción* operan o razonan en tres fases, ejecutadas por el intérprete:

1) De reconocimiento.
2) De resolución de conflictos.
3) De acción.

1) De reconocimiento. En esta fase el intérprete selecciona las *reglas* apropiadas que han sido activadas por cumplir con la condición, de acuerdo a los elementos conocidos en la base.

2) De resolución de conflictos. Si más de una *regla* ha sido seleccionada por la etapa anterior, el intérprete debe determinar qué acción va a realizar, ésto lo hace tomando en cuenta ciertos heurísticos como :

Extraer una *regla* al azar.
Tomar la *regla* que sea más específica o precisa.
Utilizar la *regla* más o la menos recientemente obtenida.
La *regla* que tenga el mayor coeficiente de verosimilitud.
De acuerdo a la prioridad de ejecución de las *reglas*.
Las *reglas* que vayan de acuerdo a un orden establecido en le base de datos.
Aplicar las *reglas* paralelamente.
Aplicar *metarreglas*.

3) De acción. Una vez elegida la *regla* a ejecutar por medio de los pasos anteriores, es aplicada, lo que da lugar al desencadenamiento de la acción correspondiente.

El procedimiento anterior se refiere al proceso de búsqueda de solución de problemas, llamado *encadenamiento hacia adelante*. Porque inicia a partir del miembro izquierdo de la regla, la condición, y termina con el derecho, la acción, si se cumple la condición.

Pero puede hacerse también de modo inverso, es decir, comenzando por el lado derecho y concluyendo con el izquierdo, a esta forma de razonamiento se le conoce como *encadenamiento hacia atrás*. En este caso las fases quedan como sigue :

1) Se eligen las reglas que concluyan en la meta que está en la parte derecha.

2) Se solucionan los conflictos encontrados.

3) Es aplicada la *regla* resultante, los elementos de las premisas resueltas se consideran como nuevos subobjetivos a alcanzar hasta que se resuelven todos los subobjetivos.

La forma de *representación del conocimiento* por medio de *reglas de producción* es muy utilizada en la **Inteligencia Artificial** porque :

- Su *sintaxis* es muy semejante a la forma de expresión humana, lo que la hace fácil de entender y de mostrar de una manera natural el *conocimiento*.
- Tiene gran modularidad, pueden añadirse, eliminarse o modificarse las *reglas* de una manera independiente.
- Hay uniformidad en la estructura del *conocimiento*, ya que se expresan las *reglas* siguiendo una forma rígida.

Entre sus desventajas se encuentran :

- Cuando se trata de representar el *conocimiento* complejo es necesario elaborar un gran número de *reglas*, incluso miles.
- Los sistemas que tienen una gran cantidad de *reglas* tienen limitaciones en su control.
- Es ineficiente porque para su ejecución se requiere la utilización de muchos recursos del sistema.
- El método de *inferencia* resulta un poco complicado, no se puede seguir con facilidad el flujo de control que lleva.

- Las *reglas* son totalmente independientes careciendo de una estructura de conjunto originando una desorganización sin guardar ninguna secuencia determinada.

R1. IF el pedido es de la zona norte THEN asignarlo al vendedor Ernesto Gómez.
R2. IF la empresa Maizoro S.A. no tiene problemas de adeudo THEN otorgarle crédito para 20 días.
R3. IF el estudio hecho de la situación financiera de la empresa Maizoro S.A. es favorable THEN R2.
R4. IF el importe del pedido es menor de $5000 THEN cobrar de contado.
R5. IF el importe del pedido es mayor o igual a $5000 AND empresa Maizoro S.A. fuera de la lista de clientes THEN R3.
R6. IF el importe del pedido es mayor o igual a $5000 AND empresa Maizoro S.A. dentro de lista de clientes THEN R2.

Figura 7. Definición de reglas de producción.

Las reglas R3, R5 y R6 son *metarreglas* porque hacen referencia a otras *reglas*.

4.4 Marcos de referencia.

Esta forma de representación fue desarrollada por Minsky en 1975, basada en el entendimiento de percepción visual y del *lenguaje natural*, aquí los nuevos *conocimientos* corresponden a conceptos que se adquieren a partir de la experiencia obtenida.

Los *marcos* son estructuras que describen objetos, situaciones o eventos.

Son un tipo de plantilla que encuadra conjuntos de *conocimientos* relacionados y relativos a un tema determinado que le da el nombre al *marco de referencia*. [Rauch 89].

El *marco* recibe un nombre (el del objeto), generalmente lo componen un conjunto de *slots* o casillas que se encargan de describir aspectos del objeto. Cada *slot* será llenado con un elemento, con un conjunto de ellos o con la referencia a otro *marco*; cuando hace referencia a otro *marco* se le llama slot liga, porque indica la relación entre dos *marcos* sólo puede apuntar cada elemento a una dirección.

El *slot* debe almacenar *conocimiento*, pero existe la posibilidad de que tenga un *conocimiento* incompleto o que carezca de *conocimiento*, además el *conocimiento* descrito en los *slots* puede ser declarativo o procedural. El *conocimiento* es declarativo cuando, el *slot* sólo hace la descripción de algún atributo del objeto referente al *marco* y procedural cuando el *slot* contiene un método o hace referencia al mismo para obtener otro elemento del *marco de referencia*.

Los *slots* se pueden llenar a través de la ejecución de procesos, pasando los valores de sus elementos.

El *conocimiento* se organiza jerárquicamente y puede darse con un alto grado de detalle.

Puesto que los *conocimientos* relacionados entre sí se agrupan conjuntamente, los *marcos de referencia*, y los sistemas basados en *marcos de referencia*, estructuran la información de un modo más organizado y manejable que los sistemas basados en *reglas de producción*. Como la estructura organizada contiene información interrelacionada estos sistemas permiten modelos más exactos de los sistemas del mundo real. Además como los *conocimientos* interrelacionados se agrupan conjuntamente, la organización en *marcos de referencia*, se asemeja más al modo en que los seres humanos recuerdan y razonan sobre el universo que les rodea[6].

[6] Minsky, escribiendo sobre su teoría de los *marcos de referencia*, dice: "Cuando encontramos una situación nueva, o cambiamos substancialmente nuestro punto de vista sobre un problema, seleccionamos de nuestra estructura un *marco de referencia*. Es decir, recordamos un *marco de referencia* que adaptamos a la realidad cambiando algunos detalles si fuese necesario". [Rauch 89].

Las características generales de los *marcos de referencia* son :

Cada objeto está asociado con un conjunto de atributos que lo describen.

Cada atributo se asocia con un valor, el cual podría tratarse de otro objeto.

El valor de los atributos puede ser :

- El nombre de un procedimiento.
- Un valor calculado durante un proceso.
- Un valor asignado.
- Un apuntador a otro objeto.
- Una constante.

Existe una jerarquía entre los *marcos de referencia*, establecida por los *slots* que apuntan hacia otros *marcos*, la jerarquía formada es taxonómica[7] ; esta situación permite la *herencia* entre los *marcos*, formando un árbol jerárquico, donde cada *marco* hereda las características del *marco* de nivel superior.

El *marco* superior es generalmente la descripción, los *marcos* padres contienen atributos que obtienen su valor de los *marcos* hijos; un marco hijo puede tener varios *marcos* padres. El *marco* que está en la parte superior de la jerarquía es el único que no tiene un *marco* padre y se le llama *marco* maestro. Si la información no se encuentra en un *marco de referencia* determinado el programa busca en la *herencia* de acuerdo con las jerarquías, es decir, un *marco de referencia* que contenga la información aplicable. Las relaciones hereditarias son importantes en la **Inteligencia Artificial** dado el enorme tamaño y complejidad de casi todos los programas. En una *relación hereditaria*, algunos objetos heredan información o atributos de otros. Una de las ventajas de las *jerarquías* de *herencia*

[7] Rauch señala al respecto que, las relaciones entre los diferentes *marcos de referencia* son taxonómicos(clasificatorios), que indican que existen características similares entre ciertos objetos de una *jerarquía*, y estas características se *heredan* de los objetos correspondientes (personas, organismos o *marcos de referencia*) en un nivel superior de la *jerarquía* taxonómica.

es la capacidad que la *jerarquía* da a los *nodos* nuevos, *marcos de referencia* u objetos del sistema para intuitivamente obtener información y significado de sus atributos, capacidades y limitaciones.

4.4.1 Jerarquía de marcos de referencia.

El tipo más importante de clasificación de los *marcos de referencia* es el jerárquico. La *relación jerárquica* es consecuencia de que los sistemas de *marcos de referencia* pueden contener pequeños *marcos* llamados submarcos de referencia, clasificados jerárquicamente.

Existen dos tipos de *marcos de referencia*: los genéricos y los específicos. Un *marcos de referencia genérico*, enumera objetos genéricos o información sobre sus ubicaciones, pero nunca nombra realmente los objetos específicos o sus ubicaciones.

Además se tienen *marcos de referencia* ejemplo(instance) que son copias singulares del *marcos de referencia* genérico de cada uno de los objetos individuales.

Para utilizar un marco, antes debe verificarse si éste puede resolver la situación actual, lo que se logra instanciando al *marco* contra la *base de conocimientos*, y se selecciona el *marco* que tenga más *slots* llenos, se prueba con la situación correspondiente y si no la cumple adecuadamente, se toma el *marco* que le sigue por el número de *slots* llenos.

De aquí resultan tres tipos de *marcos* :
El activo. El que está siendo evaluado con el problema planteado, para ser confirmado o eliminado.

El semiactivo. Aquel que integra la lista de las demás alternativas de los *marcos* que pueden llegarse a evaluar si el activo es eliminado, es el *marco* en el que aún no hay razones suficientes para su uso.

El pasivo. Cuando el *marco* no ha sido elegido para instanciarse, ya que no cumple con las características deseadas, por lo que es eliminado.

Otro aspecto de los *marcos* son los *demons*, mecanismos usados por los *marcos*, formados por un conjunto de *reglas de producción*, *proposiciones lógicas* o procedimientos contenidos en el *marco* para activarse dinámicamente cuando se accesa o deja el *marco*, se usan en caso de la necesidad de situaciones, de eliminación o adición.

Algunos han considerado a los *marcos* como una aplicación de la *programación orientada a objetos* a Inteligencia Artificial, ya que los *marcos* son muy semejantes a las *clases* de *objetos*. Los *marcos de referencia* son generalmente programados en *Lisp*.

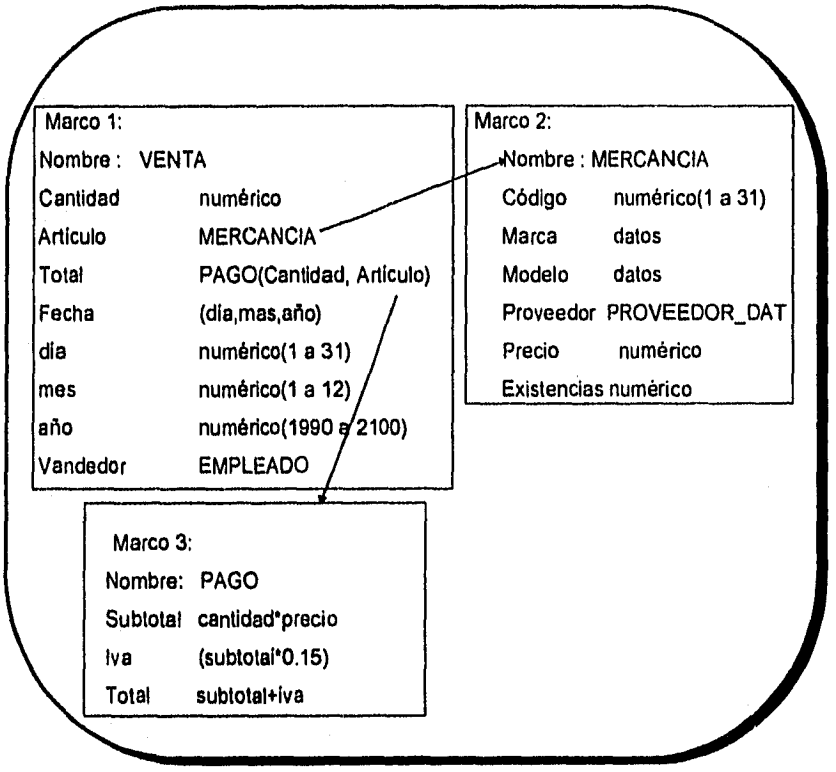


Figura 8. Ejemplos de marcos de referencia.

El *marco de referencia* 1 contiene *slots* con atributos propios del *marco* y hace el llamado a otros *marcos de referencia*.

4.5 Guiones.

Son esquemas de *representación del conocimiento* muy semejantes a los *marcos*, de los que surgen como una especialización de ellos, fueron creados por Schank y Abelson en 1975.

El *guión* es utilizado para representar una secuencia *estereotipada* de eventos aplicada a un contexto determinado. Está integrado también por un conjunto de *slots*.

Los *slots* pueden contener información acerca de los valores que podrían tener o dar un valor en caso de que ocurra una excepción o un error en donde ninguna información está disponible.

Los componentes de un *guión* son :
Condiciones de entrada. Las que deben de cumplirse para que los eventos del *guión* puedan utilizarse.

Resultados. Condiciones que serán ciertas generalmente cuando los eventos de un *guión* hayan ocurrido.

Propiedades. Son las casillas que representan a los objetos involucrados en los eventos del *guión*.

Roles. Son las casillas que representan a las personas que están involucradas en los *eventos* descritos en el *guión*.

Tanto los roles como las propiedades pueden no estar mencionados explícitamente, pero pueden inferirse.

Rutas. Dentro del *guión* existen rutas de secuencias de los eventos.

Escenas. Es una secuencia de eventos ocurridos.

Los *scripts* se ajustan a la secuencia de los eventos del mundo real como lo muestra el ejemplo de la (Figura 10).

4.6 Autómatas finitos.

Son un conjunto de estados, en donde se marca perfectamente el inicio y existe un número limitado de estados (Figura 9). Su forma de operar es determinística, es decir, para cada instrucción en particular que se ingresa, se tiene una estado de respuesta específico, no existe elección alguna. Las respuestas son determinadas sólo por la secuencia de entradas.
En los autómatas finitos existe siempre un número finito de estados y el cambio de un estado a otro es de acuerdo con las entradas de datos.

Un autómata finito es la colección de tres cosas: [Cohen 90]

1) Un conjunto finito de estados, uno de los cuales es designado como el estado inicial y otro como estado final.

2) Contiene un número Z de posibles entradas de letras que forman una cadena, de la cual se puede leer o tomar letra por letra, una a la vez.

3) Un conjunto finito de transiciones que indican a cada estado y a cada entrada qué estado será el próximo.

Los *autómatas* trabajan de manera automática, mecánica, todos sus estados están determinados, lo único que cambia son las entradas que llevan a la ejecución y cambio de estados.

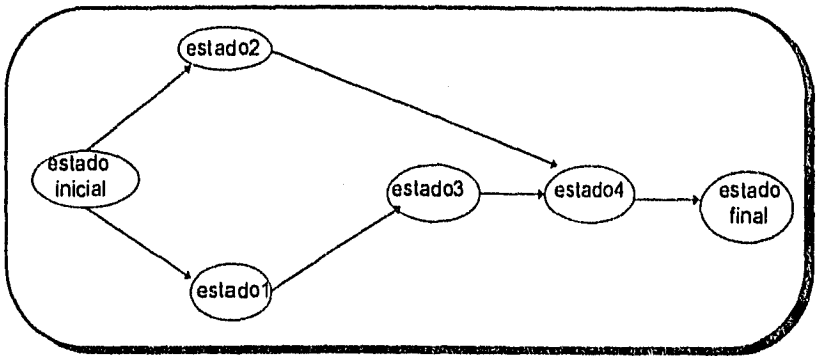


Figura 9. Autómata finito.

7.7 Programas de cómputo.

Son representados por un conjunto de instrucciones, que indican a la computadora de qué forma se realiza una cierta tarea, se expresan en un lenguaje de programación, que es rígido y sigue una serie de reglas. Semejantes a los programas escritos en lenguajes de programación como C, Pascal, Algol, etc..

Guión: Lanzamiento de un nuevo producto.

Ruta : Mercado1.

Propiedades:

Plan de Mercadotecnia.

Producto nuevo.

Muestra del producto.

Métodos de recopileción de datos.

Métodos de análisis de información.

Roles: Población.

Encuestadores.

Mercadólogos.

Ejecutivos.

Medios publicitarios.

Condiciones de Entrada:

La empresa crea un nuevo producto para introducirlo al mercado.

Resultados:

El nuevo producto es puesto a la venta para la población del Mercado 1.

Escena 1: Investigación de mercado.

Mercadólogos hacen un plan de mercadotecnia para el nuevo producto.

Encuestadores aplican métodos de recopilación de datos a un segmento de la pobleción.

Encuestadores reparten muestras del producto en el área de mercado1.

Escene 2: Decisión.

Mercadólogos aplican métodos de análisis a la información obtenida.

Mercadólogos presenten un reporte a los ejecutivos.

Ejecutivos deciden lanzar así el mercado el producto o hacer modificaciones.

Escene 3: Publicidad.

Medios publicitarios dan a conocer en forma masiva el nuevo producto a la población.

Figura 10. Declaración de un script o guión.

Capítulo III.

Búsquedas de solución de problemas en el diseño de Sistemas Inteligentes.

Capítulo III. Búsquedas de solución de problemas en el diseño de Sistemas Inteligentes.

1. Introducción.

Para llegar a concretar el diseño de los **Sistemas Inteligentes** se establece el tipo de búsqueda de solución que sea más adecuado para los objetivos del sistema, las búsquadas qua se utilizan en los **Sistamas Inteligentes** son los tipos de búsquedas que la **Intaligancia Artificial** ha desarrollado .

Para dar solución a problemas se reconocan dos tipos de búsquedas: las informales y las formales. Las primeras involucran aspectos como la intuición, el impulso, el sentido común, por lo que sólo pueden ser desarrolladas por el ser humano. Las formales, por su lado, están basadas en al análisis de situaciones o circunstancias y de información, dentro de este tipo de búsqueda se encuentran la búsqueda por optimización, la *búsqueda ciega* y la *búsqueda heurística* (Figura 11).

La búsqueda por optimización involucra el análisis numérico, se refiere a lo cuantitativo, mientras qua la *búsqueda ciega* y la *heurística* tratan el aspecto cualitativo (simbólico), aunqua también incluye al aspecto numérico.

La búsqueda por optimización espera encontrar la mejor solución al problema planteado a través de fórmulas matemáticas qua se aplicen a un modelo especifico, requiere el uso de un *algoritmo*. A este tipo de búsqueda se refieren los *algoritmos genéticos* desarrollados dentro de la **Inteligencia Artificial**.
La *búsqueda ciega* y la *heurística* son las búsquedas aplicadas por excelencia en los **Sistemas Inteligentes**, debido a que permiten manipular información de una manera más precisa.

Para algunos los términos manejados en las búsquedas de Inteligencia Artificial son más conocidos por sus nombres en inglés que en español aquí se utilizan ambas traducciones de manera indistinta, de acuerdo a la información siguiente:

Búsquedas ciegas →	Blind search.
Búsquedas heurísticas →	Heuristic search.
Búsqueda en profundidad →	Depth-first search.
Búsqueda en amplitud →	Breadth-first search.
Búsqueda de la escalada de la colina →	Hill-climbing.
Búsqueda por umbral →	Beam-search.
Búsqueda por el mejor nodo →	Best-first.
Encadenamiento hacia adelante →	Forward chaining.
Encadenamiento hacia atrás →	Backward chaining.
Retroceso →	Bactracking.

2. Búsquedas formales.

2.1 Búsqueda ciega (Blind search).

La solución de un problema se presenta como una meta a la que se llegará partiendo de condiciones iniciales y pasando por el espacio de posibles soluciones.
Las técnicas de *búsqueda ciega* exploran todas las alternativas y eventos presentados durante el proceso para encontrar la solución, tomando una a la vez. Existen dos tipos de *búsqueda ciega:* el completo o exhaustivo y el incompleto o parcial.

El tipo de búsquedas exhaustivo se da cuando son exploradas todas las alternativas existentes, encontrando diversos caminos a seguir para dar solución al problema y finalmente comparándolos para proporcionar la solución óptima, la mejor.

En la búsqueda parcial se exploran todas las alternativas encontradas antes de llegar a una primera solución, terminando en el momento que encuentra la solución, la cual puede ser buena u óptima.

La *búsqueda ciega* no resulta eficiente cuando se trata de problemas que implican diversas situaciones, porque para dar la solución va a evaluar un gran número de alternativas, ya que considera todas las que vayan surgiendo, por lo que consume ampliamente los recursos de máquina al igual que mucho tiempo. Es aplicable sólo para problemas muy simples y no utiliza ningún tipo de información especial acerca del problema, no requiere tener un conocimiento muy preciso del problema.

En los problemas complejos al aplicar esta búsqueda se produce una nueva limitación, el desbordamiento, que es resultado de la *explosión combinatoria* de posibles soluciones, es decir, que el número de alternativas crece exponencialmente, dando lugar a los problemas combinatorios, cuya característica principal es que el número de alternativas posibles se incrementa rápidamente, al ir avanzando la búsqueda se generan cada vez más alternativas haciendo más difícil su manejo.

Para dar solución a los problemas complejos se recomienda hacer uso del otro tipo de búsqueda aplicado por la Inteligencia Artificial, la *búsqueda heurística*.

2.2 Búsqueda heurística (Heuristic search).

En casos en que se cuenta con suficiente información acerca del problema puede aplicarse esta búsqueda. Por medio del conocimiento de la naturaleza y estructura del problema se diseñan métodos heurísticos, los cuales disminuyen el *espacio de búsqueda*, limitando el número de alternativas posibles.

Las reglas heurísticas se basan en la experiencia, en técnicas y conocimientos empíricos para orientar la búsqueda. Los métodos heurísticos se han desarrollado sobre bases muy sólidas puesto que implican un análisis riguroso del problema y pueden requerir en algunos casos de experimentación, por ello se considera lo suficientemente confiable para utilizarse en la **Inteligencia Artificial**.

La palabra *heurística* tiene su origen en la cultura griega, se dice que proviene de la palabra "Eureka", la cual quiere decir "lo encontré", de ésto se deriva el verbo "eurisco" que significa descubrir, la *heurística* ayuda a encontrar o descubrir una solución.

En la búsqueda de la solución de un problema en **Inteligencia Artificial**, la *heurística* fue formalizada como un conjunto de reglas utilizadas para elegir entre las diferentes alternativas presentadas, a las más apropiadas para conducir a una solución aceptable del problema.

Se recurre al uso de técnicas *heurísticas* en **Inteligencia Artificial** cuando : [Luger 89]

1) Se trata de un problema que posiblemente no tendrá una solución exacta, porque el problema en sí es ambiguo o porque los datos que se tienen no son suficientes.

2) Son problemas que pueden tener soluciones exactas, pero el costo, tiempo y esfuerzo que se necesita para encontrarlas sería excesivo, como en el caso de los problemas combinatorios, con la *búsqueda heurística* se lleva la búsqueda por el camino más prometedor, evitando hacer una revisión de todas las posibles soluciones.

La *búsqueda heurística* se considera mucho más rápida y económica que la *búsqueda ciega*.

Sus resultados son lo suficientemente buenos y en muchos casos muy cerca de las soluciones óptimas, sin embargo, las reglas heurísticas por estar basadas en la experiencia e intuición son susceptibles a fallar.

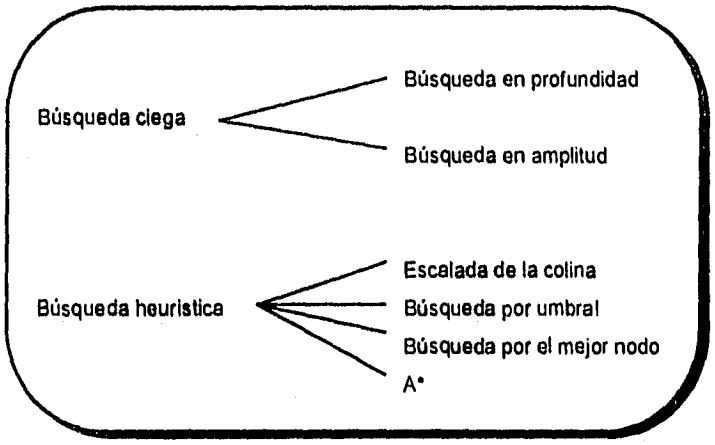


Figura 11. Diferentes tipos de búsquedas formales utilizadas en la Inteligencia Artificial.

El razonamiento seguido para llegar a la solución de un problema puede darse en dos direcciones : hacia adelante y hacia atrás, que originan los métodos de *encadenamiento hacia adelante* (*forward chaining*) y *encadenamiento hacia atrás* (*backward chaining*), considerados dentro de las *estrategias de control*.

En el *encadenamiento hacia adelante*, la búsqueda se inicia por el lado izquierdo, donde está la información con que se cuenta, los *hechos*, para llegar a obtener una conclusión que es la meta. En ésta los datos dirigen la búsqueda.

En el *encadenamiento hacia atrás*, la búsqueda es dirigida por la meta, desde donde se inicia, aquí se tratan de encontrar los *hechos* que llevan a esa conclusión.

3. Proceso de búsqueda en Inteligencia Artificial.

La Inteligencia Artificial para dar solución al problema utiliza un proceso de búsqueda que incluye tres elementos:

1) los *estados* del problema y la meta o conclusión,
2) los operadores y,
3) la *estrategia de control*

1) Los *estados* del problema, definen la situación del problema y las condiciones existentes. Los *estados* son momentáneos, son resultedo de las condiciones variantes del ambiente del problema. Los *estados* también pueden ser soluciones alternativas para el problema. Todos los *estados* son únicos.

Un *estado* es la colección de *conocimientos* disponible en forma de estructuras simbólicas. El conjunto de *estados* manejados forma el espacio de *estados*, que será examinado durante el proceso de búsqueda.

La meta es el objetivo a alcanzar, la solución o respuesta final. Un objetivo puede representar un conjunto de metas que debe cumplirse totalmente, hipótesis o precondiciones encontradas, hipótesis a ser evaluadas o simplemente explicaciones.

El conjunto de *estados* y la meta forman el *árbol* de representación de la búsqueda de solución del problema.

Los *estados* integran el *espacio de búsqueda* del problema, que se representa de manere gráfica para su mejor entendimiento con la estructura más apropiada, la de *árbol*, donde cada *nodo* es un *estado*, el *nodo* raiz es el *estado* inicial que sigue una ruta dentro del

árbol que lo lleva al estado final u objetivo, en el caso del *encadenamiento hacia adelante* y, partiendo del objetivo hacia el inicial en el *encadenamiento hacia atrás*.

2) Los operadores son procedimientos usados para pasar de un *estado* a otro. Un operador describe un proceso por el que se realiza una acción para cambiar del *estado* inicial a otro siguiente, que está de acuerdo con la *estrategia de control*, hasta que la meta es lograda. Manipulan a los elementos integrantes de la *base de conocimientos*, es decir, a los *estados* y a la meta.

3) La *estrategia de control* es la encargada de guiar los procedimientos que manipulan la *base de conocimientos*.

La *estrategia de control* se encarga de decidir cuál será la siguiente acción a realizar, qué operador se aplicará y a quién.

Si los operadores son aplicados a los *estados* para su modificación, se da un *razonamiento hacia adelante*, teniendo como objetivo que el *estado* o situación del problema sea llevado desde su configuración inicial a una condición para satisfacer la meta.

Y por el contrario, si el operador se aplica a la meta convirtiéndola en diferentes submetas más fáciles de resolver, y éstas también en otras submetas hasta que los subproblemas formados se hayan resuelto, se presenta un *razonamiento hacia atrás*, comenzando por la meta.

Las *estrategias de control* básicas son : [Turban 92]

1) *Encadenamiento hacia adelante (forward chaining).*

2) *Encadenamiento hacia atrás (backward chaining).*

3) Reducción de diferencias.

4) Al menos sucedido.

1) El *encadenamiento hacia adelante*, que ya se mencionó como forma de razonamiento lo mismo que el *encadenamiento hacia atrás*; es un método que emula la forma de razonar del hombre de manera deductiva. Corresponde a una cadena de *inferencias* que se origina en un *estado* inicial y va moviéndose hacia una meta final. Es decir, comienza con información proporcionada por el usuario y cuyos datos dirigirán la búsqueda, los aspectos del problema se van recolectando conforme se va avanzando en la solución del problema, la cual se logra al obtener la conclusión final. Es una búsqueda dirigida por los datos se la asemeja al procesamiento *bottom-up*.

Se utiliza este tipo de razonamiento cuando se tienen datos recolectados y se quieren encontrar todas las posibles conclusiones, también cuando se tiene un gran número de *estados* meta y pocos *estados* de información inicial.

2) El *encadenamiento hacia atrás*, es el método en el cual la estrategia de búsqueda es guiada por la conclusión o meta final. Tiene su inicio en la propia conclusión la que va siendo dividida en pequeñas partes para ir trabajando hacia atrás y obtener las condiciones que dieron origen a esas submetas. El proceso inicia con una hipótesis, la búsqueda trata de encontrar y verificar los *hechos* que den un soporte a esa hipótesis formulada. Este proceso termina con la aceptación o negación de la hipótesis.

Es recomendable su uso cuando se tienen pocos estados meta y muchos estados iniciales, se le conoce también como *razonamiento hacia atrás* y proceso *top-down*.

En el *razonamiento hacia adelante* lo que se representa dentro del *árbol* es el *espacio de búsqueda* de solución del problema, el cual se va ampliando al evaluar cada *nodo* y haciendo que surjan nuevos *nodos* sucesores, mientras que en el *razonamiento hacia atrás* lo que se representa en el *árbol* es la *reducción del problema*, puesto que el problema se va dividiendo en subproblemas y se hace cada vez más pequeño el problema inicial.

3) Raducción da diferancias. Es un proceso iterativo de subdividir la diferencia existenta antra el astado actual y la mata dal astado qua continúa hasta qua la difarencia as aliminada, cuando quada al astado actual como una submanta, en su conjunto sa les llaman astados intarmedios y son por los qua daba pasar la *base de conocimientos*.

Primaro determina la diferancia antra los *estados* inicial y final (meta), y selacciona el oparador que más raduca la difarancia, al aplicar al oparador sa crea un astado intarmedio. La diferancia entra al astado final y al intarmadio sa calculan, salaccionando el nuavo oparador qua más raduca la difarancia, al proceso continúa hasta qua sa determina una sacuancia da oparadores qua transforma al astado inicial en al astado mata.

Sa la llama también bidiraccional porqua aplica un conjunto da operadores que se muevan para alcanzar la mata, aqui actúa como al *razonamiento hacia adelante*. Cuando ningún oparador as aplicabla, al astado meta sa subdivida hasta que se ancuantra una submata a la qua sa la puada aplicar al operador, aquí va hacia atrás porqua astá subdividiando la mata, la búsquada tarmina cuando sa encuentran ambos razonamiantos, y se forma una ruta entra al astado inicial y al meta o final.
Sa utilizó por primara vaz en el GPS (Genaral Problam Solver).

4) Al menos sucadido. La conclusión final sa obtendrá hasta qua sa tanga suficianta información, asta estrategia se combina con la computación neuronal, la qua ayuda a sabar cuando as ya suficiente información la qua se tiana, cómo adquirir más y cómo hacer combinacionas antra éstas para obtanar más información. No es muy utilizada porqua no rasulta sancillo decidir cuando ya es suficiante información.

3.1 Retroceso (Backtracking).

Es una técnica muy utilizada durante el proceso de búsqueda dentro de la Inteligencia Artificial, por medio de la cual un programa puede retroceder desde el lugar a donde ha llegado y que ya no es posible continuar hasta el punto en donde se considera que el procedimiento llevado era correcto, se utiliza frecuentemente cuando el *conocimiento* empleado no está bien entendido, este procedimiento puede llegar a consumir mucho tiempo.

La *búsqueda en profundidad* de los métodos de *búsqueda ciega* la utiliza para poder retroceder y explorar otra rama o alternativa, así como la *búsqueda heurística, hill-climbing.*

A través del uso de esta técnica pueden elegirse otros caminos alternativos entre los que puede encontrarse la respuesta al problema, sin su utilización se llegaría a un punto o *nodo* en la búsqueda sin encontrar dicha solución convirtiéndose en búsquedas truncadas.

4. Tipos de búsquedas ciegas.

4.1 Búsqueda en profundidad (Depth-first search) .

La búsqueda se inicia en el *nodo* raíz y continúa hacia abajo hasta los niveles más inferiores del *árbol.* Un operador es aplicado a cada nodo para generar el próximo *nodo* descendiente, continuando sucesivamente hacia abajo el proceso, hasta que se encuentra la solución o ya no se pueda generar otro *nodo* teniendo que aplicar un *backtracking* para regresar al último *nodo* de decisión, es decir, aquel que está ramificado y donde se hizo la anterior elección de alternativas para elegir la ruta a seguir, y así tomar otra alternativa y realizar la búsqueda nuevamente.

Sólo se aplica el *backtracking* cuando se ha llegado a un *nodo* que no cuenta con descendientes (nodo hoja), y aún no se encuentra la solución.

El proceso de búsqueda va recorriendo el *árbol* de izquierda a derecha hasta que se encuentra la solución al problema. La *búsqueda en profundidad* garantiza el encontrar une solución, pero puede resultar muy larga, porque serán consideradas un gran número de ramas con *nodos*.

Son tomadas en cuenta todas las consecuencias posibles de cada movimiento hecho por un *estado*, antes de considerar un movimiento alternativo posible, pensando que las consecuencias se refieren a los nodos generados por otro en niveles inferiores y las alternativas a los *nodos* que se encuentran en un mismo nivel.

En la Figura 12, las flechas que van hacia abajo muestran como se recorre el *árbol* hasta su *nodo* más profundo y si la solución no se ha encontrado aún y ya no se puede bajar más por ese camino se aplica la técnica de *backtracking* para subir al *nodo* superior donde se hizo la última elección de ruta a seguir, para tomar la siguiente y continuar la búsqueda de solución.

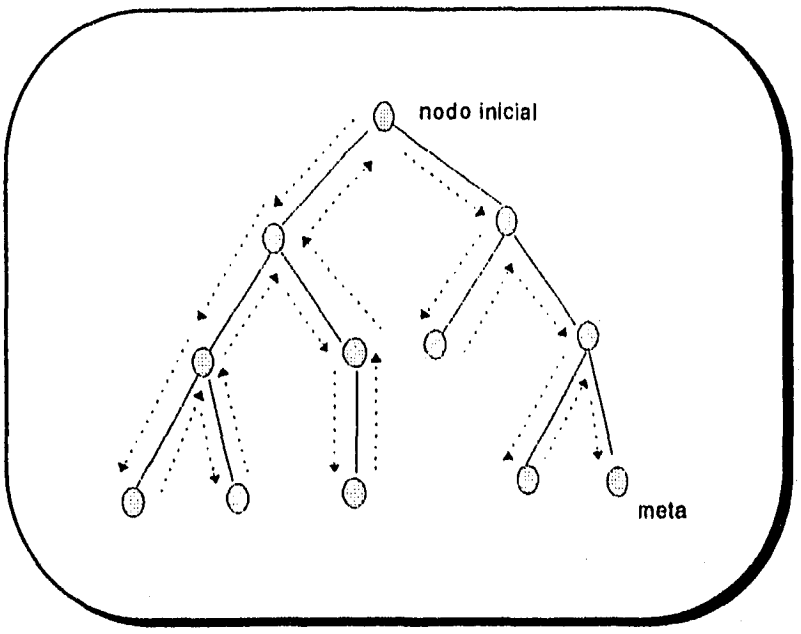


Figura 12. Representación de una búsqueda en profundidad.

4.2 Búsqueda en amplitud (Breadth-first search).

Es otro de los procesos de *búsqueda ciega*, aquí se van examinando los nodos del *árbol* por niveles, es decir, se verifican todos los *nodos* de un nivel antes de pasar al nivel inferior que le sigue; se comienza por el nodo raíz con un recorrido de izquierda a derecha.

Esta búsqueda garantiza encontrar una solución si ésta existe y el *árbol* tiene un número finito de *nodos*, aquí no se utiliza *backtracking*.

Es conveniente utilizarla cuando el número de rutas hechas es corto y donde no existe un gran número de niveles.

Sus inconvenientes son que necesita mucha memoria para almacenar todas las rutas generadas y si el número de caminos existente es amplio es muy tardado el proceso de búsqueda.

Aquí se consideran todas las alternativas posibles de cada *estado* o movimiento antes de considerar las posibles consecuencias de cada uno.

En la Figura 13 el *árbol* de búsqueda es recorrido con la *búsqueda en amplitud*, aquí las flechas indican cómo la búsqueda pasa por todos los *nodos* de cada nivel antes de buscar en alguno del nivel siguiente.

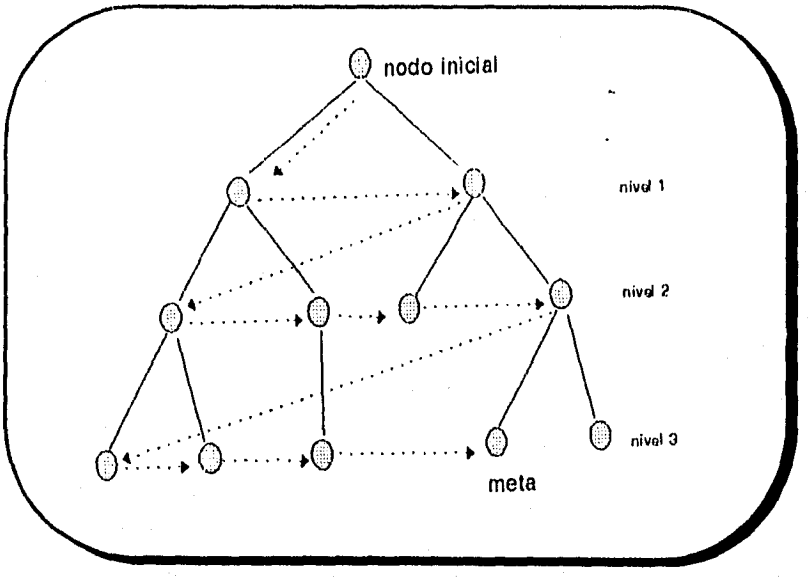


Figura 13. Representación del recorrido de la búsqueda en amplitud.

Estos dos métodos de *búsqueda ciega* no garantizan totalmente el que se llegue a encontrar la solución porque puede convertirse en una búsqueda infinita al producirse una *explosión combinatoria.*

5. Búsquedas heurísticas.

Lo que hace diferente este tipo de métodos a los de *búsqueda ciega* es que aquí se cuenta con más información acerca del problema que en los otros, que por eso reciben el nombre de *búsquedas ciegas.*

En la *búsqueda heurística* se busca por el *nodo* más "prometedor", se hace una elección entre las alternativas que ofrecen los *nodos*, la búsqueda es guiada por la selección de alternativas, la que se hará de acuerdo a las características que posea el *nodo* proporcionadas por la mayor información que se obtiene del problema.

5.1 Escalada de la colina (Hill-climbing).

Es una búsqueda similar a la de profundidad, pero aquí son elegidas por un orden de prioridad las diversas rutas a seguir, de acuerdo a los valores que se vayan obteniendo en los diferentes *nodos*, se evalúa que tan cerca está un *estado* de la meta.

A cada *nodo* se le asocia un "valor" de acuerdo a su descripción y a través de este valor se van eligiendo los *nodos* que parecen más prometedores, según la *función heurística* establecida, con el valor se estima el costo que implica el tomar la ruta de solución por ese *nodo*, los que resulten con el menor costo serán los *nodos* de mayor preferencia el camino sería por el *nodo* que tiene menos defectos.

Este proceso es más rápido que el de *búsqueda en profundidad* debido a que no va a evaluar todos los caminos ya que no explora todas las alternativas que se le presentan, sin desperdiciar tiempo en recorridos que no traerán la solución, sino que se irá sólo por los que parecen más prometedores.

Funciona como una *búsqueda en profundidad* al explorar por ramas, toma una rama prometedora y los *nodos* que resulten en esa rama son evaluados también para sólo conservar el nodo hijo que sea más óptimo, retiene sólo un *nodo*.

La búsqueda termina cuando se obtiene el objetivo o si no se encuentra a un *estado* que resulta ser mejor que cualquiera de los otros incluyendo sus hijos.

La Figura 14 correspondiente a la *búsqueda de ascensión de colinas* muestra un *árbol* que contiene *nodos* con valores obtenidos a través de una *función heurística*, que permitirá elegir el orden de los *nodos* a evaluar, no teniendo que pasar por todos los *nodos* existentes, lo que le da ventaja sobre la *búsqueda en profundidad*.

Las flechas indican la dirección de la búsqueda, dada de manera similar a la *búsqueda ciega de profundidad*, incluyendo la utilización de *backtracking*, y el orden dado por las letras es respecto a la expansión y evaluación de *nodos*.

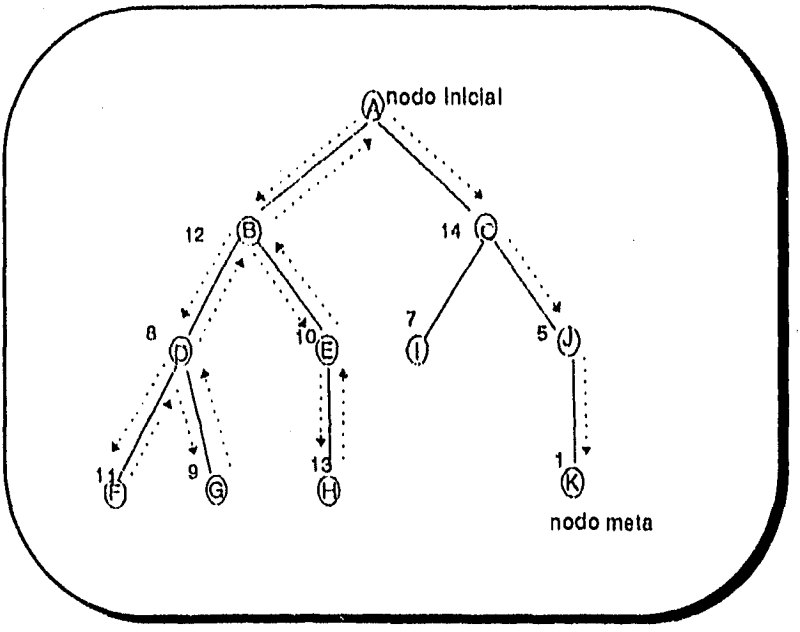


Figura 14. Representación gráfica de una búsqueda hill-climbing.

5.2 Búsqueda por umbral (Beam-search).

Este método heurístico de búsqueda elige un número M de *nodos* de un nivel, los que parecen ser más prometedores, y se va evaluando uno a la vez. Es semejante a la *búsqueda en amplitud* porque va trabajando nivel por nivel, pero no evalúa todos los *nodos* del nivel, sino que sólo los M nodos seleccionados como más prometedores. Su nombre se debe a que se mueve sólo por un sector determinado de la gráfica, la que está seleccionada o iluminada.

Es un método rápido para encontrar la solución, aunque no necesariamente sea la mejor, por lo que es heurístico.

En la Figura 15 de la *búsqueda por umbral* las flechas llevan a los *nodos* que son evaluados y en el orden establecido de acuerdo con el valor que tienen y el nivel en que están. El orden alfabético de los *nodos* es dado por el orden de expansión de los *nodos* predecesores.

Como la elección del *nodo* más prometedor es por nivel hay cierta similitud con la *búsqueda ciega de amplitud*.

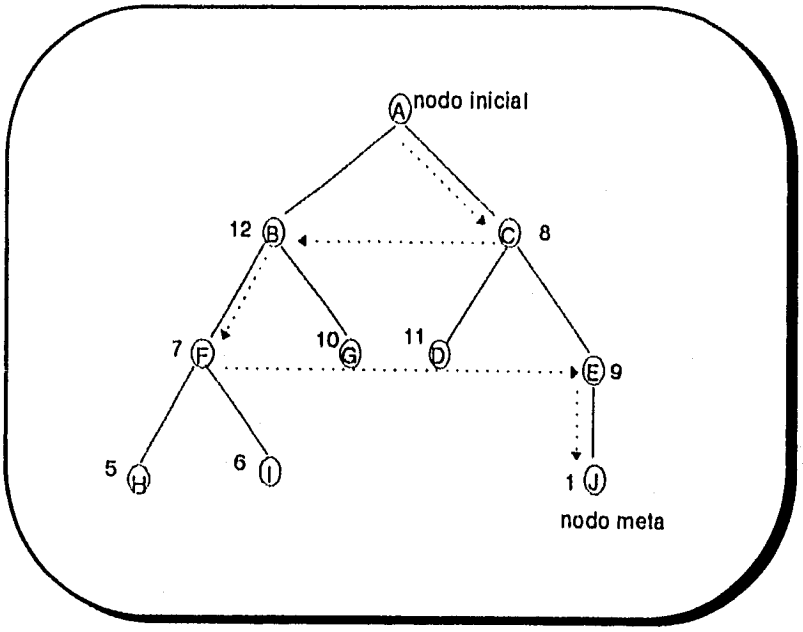


Figura 15. Representación de la búsqueda de beam-search.

5.3 Búsqueda por el mejor nodo (Best-first).

Aquí se dirige la búsqueda desde el *nodo* que resulte ser el más prometedor, no importando la posición en que se encuentre dentro del *árbol*.

Al igual que en el método *hill-climbing* aquí se puede realizar una forma semejante al *backtracking*, porque si llega a un *nodo* del que ya no hay salida se regresa a tomar otro nodo prometedor para evaluarse, pero este nodo puede estar en cualquier parte del *árbol*, no sólo en la rama que se está verificando.

Este método es una generalización de los dos anteriores, ya que va expandiendo el *árbol* hacia abajo, como el *hill-climbing*, pero si la solución no está ahí, puede tomar del mismo nivel otro *nodo* que ahora sea el más prometedor, como en el *beam-search*, para volver a evaluarlo. El combinar los métodos resulta más eficiente que cualquiera de ellos por separado.

La Figura 16 correspondiente a un árbol de búsqueda por el mejor *nodo*, inicia con la expansión del nodo raíz o inicial, que da origen a 3 nodos más (B,C,D,) que por medio de la *heurística* obtiene un valor que permite elegir al más prometedor entre ellos, se toma el *nodo* B como el más prometedor, se expande y asignan valores a sus descendientes (E,F), el *nodo* E ahora aparece como el más prometedor, a partir del nivel 2 la selección del mejor *nodo* no es sólo considerando los *nodos* de un nivel o parte del *árbol* sino que tomando en cuenta cualquier *nodo* del *árbol* que aún no se haya expandido. Al expandir el *nodo* E se obtienen *nodos* con valores superiores al *nodo* F y D, de éstos dos el más prometedor es el F, que se expande y aparece el nodo K, que tiene un valor superior al *nodo* D, por lo que se toma al *nodo* D como el más prometedor ahora, al expandirlo el *nodo* J es el mejor por lo que se expande y continúa por sus descendientes.

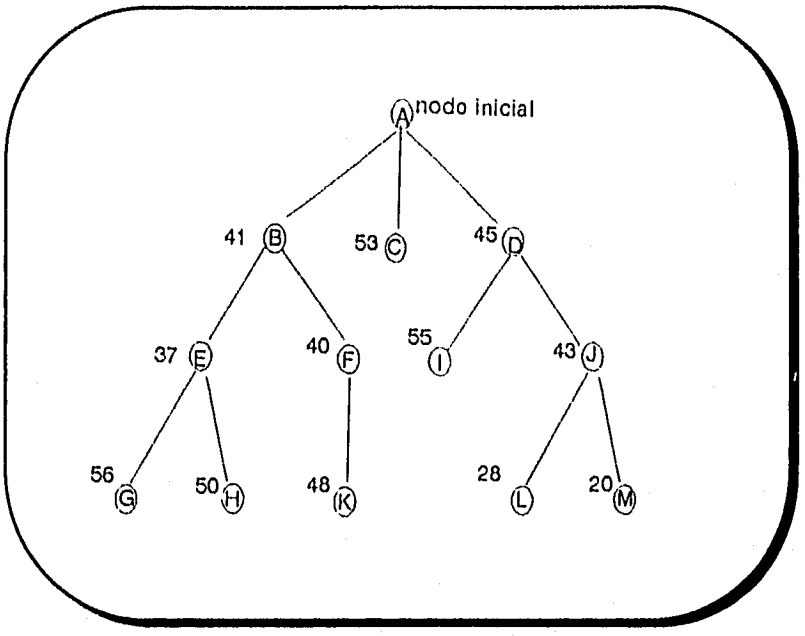


Figura 16. Representación de la búsqueda de best-first.

5.4 Algoritmo A* .

Este tipo de búsqueda heurística aplica una función de evaluación lo que permite encontrar la solución más óptima y ser la de mejor rendimiento.

Se calcula el costo del camino por el que se pasaría desde el inicio hasta llegar a la meta, f(n), eligiendo siempre los de menor costo. Este costo f(n) puede descomponerse en dos partes: una que representa el costo a partir del estado inicial hasta el nodo actual, g(n), y otra para el costo desde el nodo actual hasta el estado final o meta, h(n).

La función completa se escribe así : f(n)=g(n)+h(n).

Sin embargo, al momento de evaluar estos costos no se conoce el camino que lleva del nodo hacia el estado meta, teniendo que aplicarse entonces información *heurística.*

Los dos costos son combinados en un resultado final f(n). Los diferentes resultados de A' obtenidos se comparan y se elige el que indique el menor costo, siendo el A*.

6. Reducción del problema.

Las búsquedas anteriormente mencionadas pertenecen a las que consideran un *espacio de búsqueda* determinado, que corresponden a la utilización del *razonamiento hacia adelante,* en donde van aumentando el número de *estados* a evaluar para llegar del estado inicial hasta la meta.

El *razonamiento hacia atrás* da lugar a lo que se conoce como *reducción del problema.* Dentro de la *reducción del problema* se va dividendo a éste en subproblemas, donde cada uno de los subproblemas es más simple para resolver que el problema que lo originó.

Para la reducción del problema también se aplican las *búsquedas ciegas* y *heurísticas,* la representación de la búsqueda de solución se hace en las gráficas llamadas AND/OR, por estar compuestes por arcos que unen los *nodos* o subproblemas, llamados precisamente AND y OR.

Los arcos AND representan la necesidad de dar solución a más de un *nodo* para poder resolver el subproblema que les antecede, mientras que los arcos OR, representan aquellos donde sólo uno de ellos es suficiente de solucionar. En la gráfica los nodos AND van unidos por una línea horizontal y los OR no llevan ninguna unión.

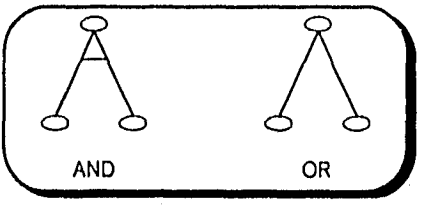


Figura 17. Arcos utilizados en la búsqueda A*.

El recorrido para la búsqueda de la solución dentro de este tipo de árboles se puede hacer empleando las dos clases de búsquedas ya mencionadas, es decir, por *búsqueda ciega* y por *búsqueda heurística.*

En la *búsqueda ciega* se emplean los dos *algoritmos*, el de *búsqueda en profundidad* y el de *búsqueda en amplitud*, siguiendo el mismo principio en cada uno como en los problemas del *espacio de búsqueda*, pero aquí en algunos *nodos* será necesario recorrerlos junto con los otros que estén unidos entre sí por un AND.

En la *búsqueda heurística* se emplea el *algoritmo* de A*, pero con una diferencia más notoria, si recordamos dentro de este *algoritmo* se utiliza una función para obtener el costo total de la ruta f(n), compuesto por la suma del costo del recorrido desde la meta hasta el *nodo* actual g(n) y, otro costo que representa la parte *heurística*, el cual indica el costo que representa ir desde ese *nodo* actual hasta la meta h(n), pero aquí no se va a utilizar el costo g(n), porque el valor de ese camino va variando conforme se avanza en el recorrido de sus *nodos* descendientes que le van aumentando el valor, lo que hace que el *nodo* que parece ser más prometedor según la función f(n) no siempre resulta así.

Este *algoritmo* se entiende mejor si revisamos un ejemplo:
En la Figura 18 del *árbol* tipo AND/OR se toma como *nodo* más prometedor a B, al expandirlo encontramos que presenta un arco AND por lo que para su solución deben resolverse esos dos *nodos* juntos, actualizando su costo ya teniendo en cuenta el de sus

descendientes tenemos que ahora vale 20, B = 10 + 8 + 2, donde D = 10, E = 8 y el 2 por tratarse de la unión de *nodos*, convirtiéndose ahora el *nodo* C en el más prometedor, cuando éste se expande se forman tres nuevos *nodos* con dos rutas expresadas con un OR y con AND que al evaluarlas resulta más apropiado tomar ese camino por el *nodo* F, tomando ahora C el valor de 13 C = 12 + 1, donde F = 12 y 1 por tratarse de un solo *nodo*, aquí vemos como el valor inicial que tenia el *nodo* es cambiado al evaluar a sus descendientes haciendo que los nodos prometedores al principio no siempre continúen siéndolo conforme se avanza.

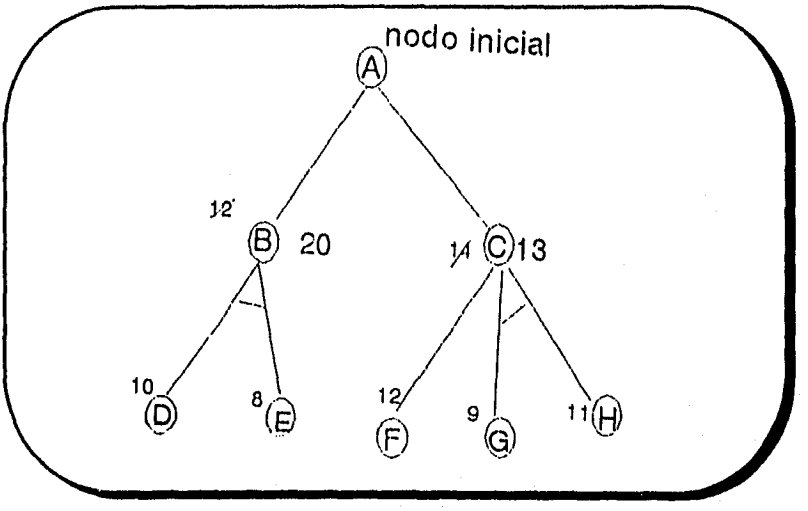


Figura 18. Árbol de búsqueda A*.

Las búsquedas más comunes dentro la solución de problemas, que destacan por su utilización son las de *búsqueda en profundidad, escalada de la colina* y *por el mejor nodo.*

Capítulo IV.

Programación Orientada a Objetos.

Capítulo IV. Programación Orientada a Objetos.

1. Introducción.

El inicio de la tecnología basada en objetos está marcada por el desarrollo del lenguaje de programación Simula, en el año de 1967, este lenguaje introduce el concepto de *objeto* en la computación. Sin embargo, es hasta finales de los años 80's cuando comienza a popularizarse este nuevo *paradigma*, se toma como una nueva alternativa para dar solución a la *crisis del software*, con la que no había podido terminar la *programación estructurada*. A finales de los 70's aparece el lenguaje Smalltalk que defino todo se define como un *objeto*, actualmente la creciente popularidad del lenguaje C++ va unida muy estrechamente e la cada vez mayor adopción de la tecnología basada en objetos por más desarrolladores de software.

Esta metodología se ocupa de modelar la realidad por medio de entidades denominadas *objetos* que interactúan entre si.

La programación tradicional consiste en procedimientos y datos, mientras que la Programación Orientada a Objetos consiste sólo de tipos de *objetos*, los cuales contienen tanto procedimientos como datos.

Los ejemplos necesarios para indicar de manera práctica los conceptos fundamentales de la Programación Orientada a Objetos será de acuerdo al lenguaje C++, ya que es lenguaje principal para esta investigación.

2. Conceptos fundamentales de la Programación Orientada a Objetos.

2.1 Objetos.

Un *objeto* es una abstracción de un conjunto de cosas del mundo real que pueden ser tanto tangibles como intangibles. Puede representar a personas, eventos, organizaciones, conceptos o a un ente cualquiera.

Estos *objetos* muestran sus *atributos* particulares y *métodos* para operarlos. Los *atributos* corresponden a los datos del *objeto*, representados por variables, que definen el estado del *objeto*; los *métodos* son las operaciones (funciones y procedimientos) que definen todo el comportamiento del *objeto*.

El estado de un *objeto* abarca todas las propiedades o características distintivas del mismo y los valores de cada una de esas. El comportamiento del *objeto* es la forma como éste actúa o reacciona a los cambios de estado por envío y recepción de *mensajes*.

Cada *objeto* tiene una identidad que lo distingue de los demás, es su identificador.

Existen objetos estáticos y dinámicos, los estáticos son aquellos que actúan sólo cuando se les requiere y los dinámicos son los que se encuentran monitoreando la ocurrencia de eventos en una aplicación y automáticamente actúan e inician las acciones, también son llamados agentes.

Algunos lenguajes de **Programación Orientada a Objetos** como C++ y Eiffel permiten el uso de objetos dinámicos, que son aquellos que van a hacer uso de la memoria de manera directa, reservando y liberando memoria. En C++ este tipo de objetos se definen a través de los operadores new y delete con los que se reserva y libera memoria, respectivamente.

Todas las *clases* de *objetos* cuentan con dos *métodos* por lo menos, que se encuentran implícitos en la *clase*: el constructor y el destructor (Figura 19), el primero es llamado cuando se quiere asignar espacio a un nuevo *objeto*, el constructor tiene el mismo nombre que la *clase* a la que pertenece, al llamarlo se crea el objeto o instancia de esa *clase*, puede existir más de un constructor en la *clase*, de acuerdo a los argumentos que lo acompañen se invocará e inicializará al *objeto*. El segundo, el destructor, es el que marca el término de un *objeto*, también tiene el nombre de la *clase* de la que forma parte el *objeto* creado, pero lo antecede el símbolo ~, esta función o *método* no regresa ningún valor ni cuenta con argumentos, sólo elimina al *objeto*.

Ambas funciones pueden ser o no declaradas por el diseñador de la *clase*, ya que están implícitas en toda *clase*.

```
//Declaración de una clase con su constructor y destructor
class empleado {
  public:
    empleado();     // función constructora de objetos de la clase.
    ~empleado();    // función destructora de los objetos de la clase.
    void registro();
  private:
    char *nombre;
    float sueldo;
};    // fin de la clase

//creación del objeto o instancia:
empleado Pedro;        // con esto se invoca al constructor de objetos de dicha clase para
 crear la instancia Pedro.
```

Figura 19. Ejemplo de la definición de una clase en C++.

Al finalizar el programa que utilice la *clase* se invocará automáticamente el destructor de *objetos* de la *clase*.

2.2 Métodos.

Son las operaciones asociadas a un *objeto*, las que modifican sus propios valores o *atributos* y que también pueden afectar a otros *objetos*.

Las operaciones básicas en todos los *objetos* son: construir y destruir, lo que marca el inicio y final de la vida del *objeto*, que se refieren como se indicó anteriormente al constructor y al destructor de *objetos*.

El *método* es un subprograma que lleva asociado argumentos que indican qué o cuánto se debe hacer con otro *método* o con un *objeto*.

Todo lo que un *objeto* puede hacer se encuentra expresado en sus *métodos*. Los *métodos* hacen que el *objeto* actúe en respuesta a un *mensaje*.

2.3 Mensajes.

Un *mensaje* es el estímulo que hace actuar al *objeto* que se encuentra recibiendo, interpretando y respondiendo constantemente a llamados de otros *objetos*.

Los *objetos* interactúan unos con otros mediante los *mensajes* que solicitan la ejecución de un *método* especifico. El *mensaje* está compuesto por el nombre del *objeto* y del *método* requeridos y si es necesario con información adicional o parámetros. El *objeto* que manda el *mensaje* se llama transmisor y el que lo recibe receptor.

Los *métodos* son los que se encargan de enviar los *mensajes* en requerimiento de una acción o información.

El conjunto de *mensajes* a los que puede responder un *objeto*, se denomina protocolo.

```
empleado Pedro;
Pedro.registro();   // mensaje que hace mención del método registro() a través de la instancia Pedro.
```

2.4 Clase.

Es un conjunto de *objetos* con estructura y comportamiento comunes. La *clase* define los *métodos* y *propiedades* semejantes en varios *objetos* de un mismo tipo en particular.

Los *objetos* que pertenecen a una *clase* se llaman *instancias* de la *clase*, las que contienen sólo los valores particulares para las variables y comparten los mismos *métodos* de sus objetos.

Los *objetos* son creados por la *clase* cuando ésta recibe un *mensaje* que expresa dicho requerimiento convirtiéndose en una *clase base* que da al nuevo *objeto* sus *métodos* y datos. El *mensaje* que es al que crea al *objeto* de la *superclase* se refiere al constructor de *objetos* de la *clase*.

Las *clases* tiene dos tipos de acceso, uno público y otro privado, la parte de acceso público se encuentran sus miembros(*métodos* y datos), que pueden ser accesados por cualquier parte del programa que haga uso de la *clase* o por otras *clases*, sean derivades o no de la propia *clase*. La parte de acceso privado es aquella que sólo pueda ser utilizada por funciones miembro de la misma *clase* y por ninguna otra de manera directa.

Cuentan además con una parte protegida en la que contiene datos y/o funciones miembro que comparte con las *clases derivadas*.

2.5 Herencia.

Es un mecanismo que permite compartir *métodos* y propiedades entre *clases*. La *clase base* que contiene la definición de estos *métodos* y propiedades se denomina *superclase*, de la cual se derivan otras *clases* que reciben el nombre de *subclases*, la *superclase* hereda sus *métodos* y valores a las *subclases*.

La *subclase* define el comportamiento de *objetos* que heredan algunas características de la clase padre o base, pero también añade características especiales no compartidas por la *clase base* por lo que la *subclase* es una especialización de la *superclase*.

Existen dos tipos de *herencia*: la simple y la múltiple. La primera se da cuando una *subclase* puede heredar de una sola *clase* y la segunda se refiere a la habilidad de una *clase* de adquirir *métodos* y *propiedades* de más de una *clase*.

La **Programación Orientada a Objetos** introdujo el concepto de *herencia*, que es su principal contribución ya que permite lograr la *reusabilidad de código*.

La *herencia* da lugar a la *jerarquía* entre *clases*, establecida por la subordinación de las *subclases* a la *superclase*.

Las *clases derivadas* pueden acceder de manera directa sólo a las partes pública y protegida de la *clase base* (Cuadro 5).

Tipo de acceso	Acceso por la propia clase	Acceso por clases derivadas	Acceso por objetos de otras clases
público	Si	Si	Si
privado	Si	No	No
protegido	Si	Si	No

Cuadro 5. Acceso a los miembros de una clase base.

```
// la clase obrero es derivada de la clase base empleado.

class obrero: public empleado{
        public:
                // parte publica de la clase derivada
        private:
                // parte privada de la clase derivada
        protected:
                // parte protegida de la clase derivada
  } // fin de la clase derivada
```

Figura 20. Clase derivada.

2.6 Abstracción.

Es la separación de detalles innecesarios en la especificación de un sistema, haciendo énfasis sólo en algunos detalles o propiedades y suprimiendo otros que son irrelevantes.

Los *objetos* pueden abstraerse de diversas maneras dependiendo del observador.

De la abstracción se deriva qué *objetos* se requieren para la realización del sistema, se especifican además los datos y funciones que formarán parte de cada *objeto*.

2.7 Encapsulamiento.

Es el ocultamiento de información que permite separar al *objeto* de los aspectos externos al mismo, haciendo que nadie ajeno al propio *objeto* puede accesar a sus datos y/o *métodos* miembros, es decir, que sólo el *objeto* a través de sus *métodos* tiene permitido manipular ciertos valores.

Los programas que usan *objetos* sólo pueden conocer de ellos lo que los propios *objetos* les hacen público.

2.8 Polimorfismo.

Esta característica de los *objetos* permite tener diferentes *métodos* con un mismo nombre. Tanto la *superclase* como las *subclases* pueden definir *métodos* con acciones diferentes para un identificador común.

```
class obrero: public empleado {
        public:
                registro();          // esta función miembro es redeclarada por la función derivada
                proceso();
} // fin de la clase derivada
```

Figura 21. Polimorfismo de funciones.

Dentro de una misma *clase* también puede darse el *polimorfismo*, redeclarar más de una vez una de sus funciones, la manera en que se diferencian es por el número de sus argumentos, el compilador es el encargado de decidir qué *método* o funciones debe utilizar de acuerdo con el tipo de argumentos invocados.

La sobrecarga de operadores es también un ejemplo de *polimorfismo*, ya que se redefine el uso de los operadores definidos ya por el propio lenguaje. Es común su uso para declarar las operaciones entre números complejos con los operadores aritméticos de suma, resta, multiplicación y división.

2.9 Persistencia.

Se refiere a la permanencia de un *objeto*, es decir, al tiempo en el cual el *objeto* es puesto en memoria y se hace accesible. Cuando un *objeto* no se utiliza por largo tiempo es destruido y liberado su espacio de memoria para reutilizarse. Los *objetos* son de corta *persistencia* en memoria.

Además pueden guardarse los *objetos* en dispositivos de almacenamiento secundario, para hacerlos permanentes, ésto se haría de manera opcional, no lo ofrece directamente el lenguaje de *objetos*, depende del programador.

Lo más recomendable es no guardar los *objetos* por mucho tiempo, sino crearlos de nuevo cada vez, sólo guardar los datos necesarios para su creación en algún archivo.

3. Lenguajes principales de la Programación Orientada a Objetos.

3.1 Smalltalk.

Es un lenguaje de **Programación Orientada a Objetos** nacido en los laboratorios de Xerox, bajo la dirección de Alan Kay y Adele Goldberg, su propósito final era crear una máquina pequeña con la capacidad de manejar ambientes gráficos como las computadoras personales de la actualidad, llamada Dynabook, para desarrollar esa idea pensaron en un lenguaje semejante al primer lenguaje para **Programación Orientada a Objetos**, Simula 67, así crean la primera versión de Smalltalk en 1972 y que culminó en la versión final de Smalltalk-80. En este nuevo lenguaje se manejan todos los datos como *objetos*, Smalltalk es un lenguaje de programación con entorno de programación integrados, introduce conceptos ahora muy utilizados como el manejo de menús, ratón, ventanas, iconos y gráficas en general, a través de su desarrollo se hizo más fácil el manejo de las computadoras para los usuarios, sin embargo, en su inicio las máquinas con que se contaba no hacian posible que estas innovaciones llegaran a todos los usuarios, ya que requerian gran capacidad en memoria y resolución en los monitores, por lo que sólo pudo instalarse en equipos grandes y las estaciones de trabajo que podían utilizarla eran muy costosas, Apple fue el primero en implementar un sistema basado en Smalltalk en su máquina Lisa, pero ésto la hacia muy costosa, después basado en este primer desarrollo crea su máquina Macintosh, la cual es recibida con agrado por el público en general, con lo que Smalltalk es aceptado como un lenguaje que facilita el uso de las computadoras.

Las ideas principales que dieron origen a Smalltalk se enfocan a la idea de Kay y Goldberg acerca de cómo debería ser la computación, en cuanto a:
Interacción, referente a que el entorno de computación debe hacer lo posible para proporcionar una retroalimentación inmediata para todas las acciones del usuario.

Gráficos, las personas tienen la facilidad para interpretar formatos gráficos por lo que debe de poderse sustituir al máximo el texto por gráficos.

Programación Orientada a Objetos, haciendo posible que la máquina opere con términos más cercanos a la realidad del usuario.

En Smalltalk los datos de los objetos son privados y sólo es posible accederlos por medio de sus procedimientos, éstos si aparecen como públicos, los procedimientos mediante *mensajes* manejan a los *objetos*.

La versión de Smalltalk V maneja memoria virtual, cuando ya no es suficiente la memoria con que cuenta el sistema, el programa ocupa inmediatamente parte no utilizada del disco y allí coloca los *objetos* nuevos, creando memoria artificial y hace posible la persistencia del programa.

3.2 C++.

Es el lenguaje de **Programación Orientada a Objetos** basado en el lenguaje C, fue desarrollado en 1980 por Bjarne Stroustrup de Laboratorios Bell, su nombre se debe a que es concebido como un paso adelante de su lenguaje base. Debido a que maneja la forma clásica de C con adición de los conceptos de **Programación Orientada a Objetos** se le considera un lenguaje híbrido.

Con C++ se pretende mejorar la calidad del *software*, hacer reutilización de código y facilidad de adaptación en los sistemas grandes.
Los objetivos de C++ son:
Mantener la eficiencia y transportabilidad de C.
Hacerlo compatible con las versiones anteriores de C.
Actualizar a C al *paradigma* de **Programación Orientada a Objetos** para producir *software* de calidad.

Los tipos de datos que pueden ser utilizados en los programas de C++ son:

Tipo	Tamaño en bytes	Rango
char	1	-128 a 127
unsigned char	1	0 a 255
signed char	1	-128 a 127
short	2	-32768 a 32767
unsigned short	2	0 a 65535
int	2	-32768 a 32767
unsigned int	2	0 a 65535
long	4	± 2.1*10^9
unsigned long	4	0 a 4.2*10^9
float	4	± 3.4*10^23
double	8	± 1.7*10^308
long double	10	±3.4*10^4932 a 1.2*10^4932

Cuadro 6. Tipos de datos en C++.

La estructura de un programa escrito en C++ es la misma que la definida por ANSI C, la que contiene en primer lugar las directivas al preprocesador, las cuales incluyen los archivos de cabecera #include así como #define, la función principal main(), que contiene variables e instrucciones y finalmente pueden agregarse también otras funciones que utiliza la función principal main(). Todo programa escrito en C++ para poder ser ejecutado requiere contar con la función principal main().

```
#include   //directivas
#define
void main(){          // función principal
 intrucciones...;              //cuerpo de la función main(),
}
void func1(){}        // otras funciones()
```

Figura 22. Esquema de un programa en C++.

Las instrucciones contenidas dentro de las funciones pueden ser de tres tipos:

1)De expresión, las cuales indican una secuencia simple del flujo de datos como: a+b; a=2.345;

2) De iteración o ciclo como :

```
do{
  instrucciones;
}while(condición);

for(inicialización,condición,incremento) {
  instrucciones;
};

while(condición) {
 instrucciones;
};
```

3)De decisión:

```
if(condición)
 instrucción;
else;
 instrucción;

switch(expresión) {
    case 1: instrucción;
            break;
  ...case n: instrucción;
            break
};
```

Los operadores son los mismos que se utilizan en el lenguaje ANSI C. C++ utiliza para la lectura y escritura de datos las funciones ya empleadas en ANSI C como scanf, printf, getche(),puts, entre otras, sin embargo, en las versiones de C++ se añaden nuevas funciones para realizar estas operaciones, las principales son cout<< y cin>>, con las que ya no se requiere indicar el tipo de dato que será manipulado como sucedia con las funciones previas.

Es un lenguaje muy utilizado actualmente por los desarrolladores de sistemas basados en *objetos*, varios fabricantes han sacado al mercado el lenguaje como Borland, Microsoft, Symantec e IBM, para plataformas que van desde computadoras personales hasta máquinas grandes.

3.3 Eiffel.

Es un lenguaje de Programación Orientada a Objetos creado por Bertrand Meyer en 1985, fue creado no sólo como un lenguaje más de Programación Orientada a Objetos, sino como una nueva manera de conceptualizar, diseñar e implementar el *software* orientado a *objetos*, ya que incluye una metodología para modelar el sistema a realizar a través de la Programación Orientada a Objetos con el objetivo de crear software de calidad.

Es considerado por su creador como un lenguaje puro de Programación Orientada a Objetos con un poderoso ambiente gráfico.

Eiffel hace un manejo automático de liberación de memoria para su reutilización cuando los *objetos* no son usados por un espacio determinado de tiempo, lo que se reciba el nombre de manejo de basura o recolección de basura.

A pesar del tiempo que ya tiene en el mercado Eiffel, aún no ha podido convertirse en un lenguaje popular dentro de los programadores de *objetos*, su versión más actual es Eiffel 3.0.

4. Relación entre Inteligencia Artificial y Programación Orientada a Objetos.

Cuando se trabaja con *marcos de referencia* o con *guiones*, los programadores de Inteligencia Artificial, desarrollan ambientes similares a los creados por la Programación Orientada a Objetos. Aquí se lleva a cabo la estructuración de *objetos* conceptuales manipulados en cierta actividad.

Tanto los *marcos de referencia* como los *guiones* se agrupan de manera estructurada con el conjunto de *conocimientos* relativos a un *objeto* físico, un concepto o una situación.

El *marco de referencia* está integrado por un conjunto de *atributos* colocados en *ranuras* o *slots* del mismo modo que los *objetos* comunes de la Programación Orientada a Objetos. Los *guiones* también se basan en el mismo principio de representación que los *marcos*, pero éstos se refieren a la descripción de secuencias de eventos o acontecimientos y no a objetos estáticos.

Cada *objeto* corresponde a un *marco de referencia*, que además de contar con sus *atributos* tiene también como todo *objeto* una serie de *métodos* con los que realizará la manipulación de *objetos*. Estos *objetos* son independientes y pueden comunicarse entre sí por medio de *mensajes*.

La *herencia* de propiedades propicia la transmisión de *conocimientos* de un *objeto* hacia sus descendientes.

El razonamiento se logra al comparar una *instancia* con una *clase*, a través de un mecanismo iterativo, dándose una interpretación de los *objetos*.

Estas formas de representación son muy útiles en los *sistemas expertos*, en *visión por computadora* y en el procesamiento del *lenguaje natural*.

Las representaciones por *objetos* en **Inteligencia Artificial** permiten la abstracción de un dominio reuniendo en una estructura el conjunto de elementos de *conocimientos* relativos a ese concepto, incluye tanto la forma de *representación del conocimiento procedural* como la declarativa y muestra una jerarquización de los *conocimientos* y razonamientos.

4.1 Integración de los dos paradigmas.

Las herramientas que se han hecho aplicando la **Programación Orientada a Objetos** a la **Inteligencia Artificial** se basan principalmente en el lenguaje Smalltalk, las versiones de *Prolog* en las que aparece utilizando también Smalltalk, son las más recurridas por las universidades que están desarrollando estos híbridos. Los desarrollos se basan en este lenguaje de **Programación Orientada a Objetos** principalmente, por haber sido el primero en aparecer dentro de la **Programación Orientada a Objetos** y porque se desarrolló especialmente para la misma, en cambio C++ es una adaptación del lenguaje C a la **Programación Orientada a Objetos**.

Los desarrollos creados uniendo estos dos *paradigmas* de la computación aún no se comercializan plenamente, ya que continúan en fase de pruebas.

La *programación lógica* ha sido la elegida para aprovecharla con la **Programación Orientada a Objetos** y como *Prolog* es el lenguaje representante de este tipo de programación es del que se han creado diversas versiones que incluyen las características de la **Programación Orientada a Objetos**, entre sus versiones están OL(P), L&O. Logtalk, Prolog++, Nuoo-Prolog, Object Pro.
Del lenguaje *Lisp* también se han hecho algunas versiones en las que aparecen características de la **Programación Orientada a Objetos** como CLOS y LOOPS.

Otros lenguajes creados combinando la *programación lógica* y la **Programación Orientada a Objetos** son :
BeBop, Life, Rock & Roll, Flogic & Transmition Logic Prototypes, DLP, Oriented84/k, Bridge, KSL/logic, LogicC++, I++, G y Uniform.

Estos lenguajes creados han tomado de la **Programación Orientada a Objetos** sus características principales de *encapsulamiento, polimorfismo* y *herencia.*

A través de la *programación lógica* de la **Inteligencia Artificial** en combinación con la **Programación Orientada a Objetos** se une la programación de un lenguaje declarativo, como *Prolog*, y un lenguaje de tipo procedural como Smalltalk y C++.

En la *programación lógica* se resuelven los problemas de una manera declarative, en la que no importa el uso que se hará del *conocimiento* almacenado, sino sólo se trata de describir qué *conocimiento* será utilizado, permitiendo realizar aplicaciones por medio de grandes *bases de conocimientos*; a la **Programación Orientada a Objetos** por ser procedural le toca hacer la parte referente a la utilización de ese *conocimiento*, los beneficios que ofrecen son el poder modelar aspectos relevantes del mundo real considerándolos como *objetos* autónomos que pueden encapsular sus datos o *atributos* y procedimientos o *métodos*, estos *objetos* se estructuran de forma jerárquica permitiendo el manejo de la *herencia* y apoyando la *reusabilidad de código* y el mantenimiento de los sistemas creados.

Los *objetos* son una forma que resulta adecuada para la representación de estructuras de *conocimientos* así como de datos en general.

La *herencia* dentro de los *objetos* permite definir información en el nivel superior más adecuado y transmitirla a sus descendientes de acuerdo con la *taxonomía.*

4.2 Lenguajes y herramientas ya creados utilizando Programación Orientada a Objetos e Inteligencia Artificial.

Prolog++.

Es un lenguaje que combina lo mejor de la Inteligencia Artificial y de la Programación Orientada a Objetos, es Programación Orientada a Objetos integrada con el lenguaje *Prolog*, fue creado en 1989 por Logic Programming Associates. Sus desarrolladores consideraron que los *objetos* proveen con una forma conveniente la estructuración de las relaciones entre *conocimientos* y datos. Existe la posibilidad de definir *taxonomías* de *objetos* con Prolog++ y de manipularlos con las reglas dadas por *Prolog*.

Este lenguaje hereda las características de programación declarativa para resolver los problemas y las de las *bases de conocimientos* de *Prolog* y las características principales de la Programación Orientada a Objetos.

Las características tomadas de la Programación Orientada a Objetos son:

- La *jerarquía* de *clases* y la *herencia*, a través de esta *jerarquía* se establecen los diferentes niveles entre las *clases* indicando quién es la *clase base* o *superclase* y quiénes son las *clases derivadas* o *subclases*. La *jerarquía* y la *herencia* están muy ligadas puesto que, cada una depende de la otra, pudiendo heredarse de una *clase* superior a una descendiente sus *métodos* y *atributos*. Prolog++ soporta un tipo de *herencia múltiple*, es decir, que cada *subclase* puede heredar de más de una *clase* superior.

- Objetos estáticos y dinámicos, Prolog++ soporta ambos tipos de *objetos*, los estáticos se refieren a aquellos que se crean desde el momento de la compilación del programa, mientras que los dinámicos se van creando durante su ejecución al ser requeridos.

- *Polimorfismo*, Prolog++ puede usar diferentes *métodos* en distintas *clases* por medio de un mismo *mensaje*, dabido a que diversas *clases* pueden contener *métodos* a los que se hace referencia de manera similar.

- *Encapsulamiento* y *abstracción*, todas las dafiniciones de *clases* están delimitadas por su inicio y fin, pudiendo ancarrar cada una tanto sus *métodos* como *atributos*. Los *métodos* pueden ser dafinidos de manera pública o privada, como en la **Programación Orientada a Objetos**.

Los dasarrollos hechos en Prolog++ puedan sar integrados con los programas de *Prolog* estándar, las *clases* dentro da Prolog++ puedan hacer uso de los *predicados* de *Prolog*, Prolog++ tiene acceso directo a *Prolog*. Se han desarrollado versiones de Prolog++ para ambientes Windows, MS-DOS y Macintosh.

Logtalk.

Su nombre se deriva de **Prolog+Smalltalk=Logtalk.**

Se hizo también con el objetivo de unir las ventajas de la **Programación Orientada a Objetos** y de la **Inteligencia Artificial**, tomando en cuanta que la **Programación Orientada a Objetos** les permite trabajar con las mismas entidades en diversas fases dal desarrollo y la aplicación, así como también el podar organizar y encapsular el *conocimiento* de cada entidad.

La **Inteligencia Artificial** a través de la *programación lógica* permite represantar de forma declarativa al *conocimiento* de cada entidad, uniendo ambas ventajas que proporcionan estas dos formas de programación se busca reducir la distancia entre la aplicación y la solución, majorando en la escritura y mantenimiento de las aplicaciones.

Logtalk es compatible con la mayoría de los compiladores de *Prolog*, ya que se basa en los estándares de *Prolog* dados por ISO[5] , pudiendo ejecutar sus programas en tales compiladores.

Los *objetos* son tomados para reflejar el *conocimiento* que puede tenerse de cada entidad modelada, es decir, el *conocimiento* es todo lo que concierne al *objeto*, lo que se puede decir acerca del *objeto*. A los *objetos* los toman como una teoría, que pertenece a una *clase* representada como metateoría. Introduce un concepto nuevo de *clases*, las metaclases, que hace referencia a la *clase* de *clases*, aquí los *objetos* son considerados *clases* que pueden trabajarse de manera independiente.

El único tipo de entidad soportado por Logtalk son los *objetos*, pero éstos como ya se dijo son considerados *clases* y de acuerdo a como se definan pueden ser *clases*, *subclases*, *superclases* o metaclases.

La *herencia* que se puede establecer es del tipo simple, debido a que une *clase* sólo puede ser la especialización de una *superclase*.

LIFE.

EL nombre de este lenguaje está formado de la primera letra correspondiente a : Logic, Inheritance, Functions and Equations.

Es un lenguaje de programación para la **Inteligencia Artificial**, ya que puede manipular *símbolos*. Combina como su nombre lo indica la programación simbólica y la **Programación Orientada a Objetos**, es también una versión que surge de los principios de *Prolog*, haciendo uso de la *representación declarativa*.

[5] ISO=International Standard Organization, Organización de estándares internacionales.

Quintus Objects.

Fue creado como una opción para que los programadores de *Prolog* puedan definir *objetos* dentro de sus programas. Los *objetos* son estructuras de datos que mantienen información y responden a comandos que representan a los *mensajes*. Cada tipo de *objeto* es tomado como una *clase*. Los *objetos* cuentan con ranuras en donde se mantienen los datos, las *ranuras* pueden ser de tipo entero, caracter, string, flotantes o términos de *Prolog*.

Los *métodos* por los cuales el *objeto* responde a los *mensajes* son definidos como cláusulas de *Prolog*.

Quintus Objects maneja la característica principal de la **Programación Orientada a Objetos**, la *herencia*, cuando una nueva *clase* es definida como hija de otra, ésta hereda les ranuras y *métodos* de su *clase* predecesora, además de poder definir los propios o redefinir los de la *clase*, dándose también el *polimorfismo*, la *herencia* manejada puede ser múltiple, heredando una *clase* de más de una anterior.

Permite interactuar con *Prolog* a través de la creación y envío de *mensajes* a *objetos* por los programas de *Prolog*.

Oz.

Es un lenguaje de programación diseñado para *programación lógica*, se desarrollo como sucesor de los principales lenguajes de Inteligencia Artificial, es decir, de *Prolog* y de *Lisp*, combinándolos con Smalltalk.

Como los demás lenguajes toma de la Programación Orientada a Objetos el concepto de *herencia*, manejada como herencia múltiple, los *objetos* son sus estructuras básicas, utiliza la *encapsulación* de datos, las *clases* y *objetos* se crean dinámicamente.

A diferencia del funcionamiento de *Prolog* no utiliza la estrategia de control de *backtracking*, hace una búsqueda en un espacio menor del que tomaria *Prolog*.

CLOS.

CLOS=Common Lisp Object System.

Se trata de la versión estandarizada de *Lisp* para **Programación Orientada a Objetos**, por el lado de la **Programación Orientada a Objetos** se basa en Smalltalk; permite el manejo de *herencia* múltiple, permitiendo además resolver la ambigüedad que pueda surgir. Utiliza argumentos múltiples para resolver los *métodos*, con lo que puede implementar operaciones de *polimorfismo* múltiple, en el que depende de varios argumentos la implementación de un *método*. No cuenta con verificación de tipos, de los que tiene poca variación.

Este lenguaje ha sido utilizado para la creación de *bases de conocimiento* para *sistemas expertos* para lo que se dice que es una alternativa sencilla.

CLOS permite un poderoso manejo de la *herencia*, dando lugar a la *herencia* múltiple. Permite también la construcción da métodos nuevos durante la ejecución.

El lenguaje declara inicialmente como de acceso público todo lo definido por las *clases*, para proteger a través del encapsulado a sus miembros es necesario que el programador de la *clase* lo indique explícitamente.

Capítulo V.

Programación de las directivas de C++ Inteligente.

Capítulo V. Programación de las directivas de C++ Inteligente.

1. Introducción.

Les búsquedes de solución de problemes junto con la *representación de conocimientos* son los elementos principales de la **Inteligencia Artificial**, los **Sistemas Inteligentes** toman de equelle estos elementos, cuya definición forma parte de las etapas de análisis y diseño de la ingeniería de *software* para **Sistemas Inteligentes**.

Para la progremación de **Sistemas Inteligentes** a través de C++ Inteligente es importante aclarar que la *representación del conocimiento* que se utiliza es mixta, semejante a la de los *marcos de referencia*, ya que incluye los datos y *métodos* de búsqueda con que se resolverá el problema presentado.

En este capítulo se muestran las búsquedas inteligentes como el principal elemento a programar, creando las directivas #include, que contienen tanto las *búsquedas ciegas* como las *búsquedas heurísticas*, por medio de las cuales se manipula la *base de conocimientos* para llegar e la solución del problema.

Se da un prototipo de los elementos básicos que deben incluirse para realizar las búsquedas inteligentes. Pare verificar la aplicación de estos archivos de cabecera se muestra un programa que utiliza las diferentes búsquedas inteligentes, con las que puede realizarse un **Sistema Inteligente**. La dirección del razonamiento en que se basan las búsquedas es hacia adelente (*forward chaining*).

A treves de la realización de estas directivas se hace la inclusión de las mismas en el lenguaje C++, con lo que se de el inicio de una nueva modelidad de C++, el C++ Inteligente.

En capítulos anteriores ya se ha hecho una exposición amplia acerca de las búsquedas de solución, pero para comprender mejor su programación es este capítulo se incluye un resumen y los *algoritmos* de cada una de las búsquedas utilizadas.

2. Algoritmos de búsquedas de solución en Inteligencia Artificial.

2.1 Categorías de clasificación de los algoritmos de búsquedas Inteligentes.

Los *algoritmos* de búsqueda utilizados dentro de la Inteligencia Artificial para dar solución a problemas se encuentra divididos en 2 categorías:

Los representativos de las *búsquedas ciegas* y los de *búsquedas heurísticas.*

Las características principales de ambas categorías pueden expresarse como se muestran en el siguiente cuadro:

Búsqueda heurística	Búsqueda ciega
Sólo considera los estados que parecen ser más prometedores para llegar a la solución	Checa todos los *estados de búsqueda* para encontrar la solución.
Termina cuando encuentra la primera solución lo suficientemente buena.	Puede terminar cuando haya encontrado todas las soluciones posibles y elegir la mejor o cuando encuentre la primera solución.
La solución al problema encontrada no siempre es la mejor.	Puede dar la solución más óptima al problema.
Requiere más información acerca del problema a resolver.	La información que utiliza para la solución del problema es escasa.
Es poco costosa en cuanto al uso de recursos de computadora.	Puede resultar muy costosa en la utilización de recursos.

Cuadro 7. Características de búsquedas en Inteligencia Artificial.

Los *algoritmos* de búsqueda operan sobre los estados que integran el espacio de búsqueda de solución del problema, estos estados representan las diferentes alternativas a través de las se realizarán las evaluaciones necesarias, del espacio de búsqueda que forman surgirá la ruta que lleva a la solución.

2.2 Algoritmos generales de búsquedas ciegas.

2.2.1 Búsqueda primero en profundidad.

1. Generar estado.
2. Si estado=objetivo entonces termina encontró solución.
3. Genera descendientes.
4. Si descendientes>0 entonces estado=descendiente sin revisar ir a 2.
5. Hacer backtracking.
6. Si encontró estado ir a 2.

Figura 23. Algoritmo de búsqueda en profundidad.

Esta búsqueda hece uso de la estrategia de *backtracking*, a través de la cual regresa desde un estado que no es solución y del que ya no hay más alternativas, hasta el que le anteceda que cuente con más alternativas a evaluar, el recorrido vertical, ya que va descendiendo a través de una rama.

2.2.1.1 Algoritmo para backtracking1.

1. Regresa a estado predecesor.
2. No existe estado predecesor termina y regresa valor nulo.
3. No tiene descendientes sin revisar ir a 1.
4. Estado=descendiente+1 sin revisar.
5. Termina y regresa estado encontrado.

Figura 24. Algoritmo de backtracking1 para la búsqueda en profundidad.

Este *algoritmo* de *backtracking* se ocupa de regresar por la rama explorada hasta el estado que cuenta con más descendientes sin revisar aún, tomando el siguiente estado descendiente alternativo que se encuentre a la derecha del antes revisado.

2.2.2 Búsqueda en amplitud.

1. Genera estado en nivel n.
2. Si estado=objetivo entonces termina encontró solución.
3. Genera descendientes.
4. Si mas estados en nivel N ir a 2.
5. Cambia a nivel N+1.
6. Si número de estados=0 en nivel N termina no encontró solución.
7. Toma estado ir a 2.

Figura 25. Algoritmo de búsqueda en amplitud

Esta *búsqueda ciega* va evaluando los estados nivel por nivel, el cambio de nivel se hace hasta que se hayan revisado todos los estados del nivel presente, no requiere utilizar *backtracking*.

2.3 Algoritmos de búsquedas heurísticas.

2.3.1 Algoritmo de la búsqueda escalada de la colina.

1. Generar estado.
2. Si estado=objetivo entonces termina encontró solución.
3. Genera descendientes.
4. Si descendientes>0 entonces buscar descendiente más prometedor.
5. Estado=descendiente más prometedor sin revisar, Ir a 2.
6. Hacer backtraking2.
7. Si estado=nulo termina no encontró solución.
8. Ir a 2.

Figura 26. Algoritmo de búsqueda escalada de la colina.

Esta *búsqueda heurística* trabaja de manera similar a la *búsqueda ciega* de *primero en profundidad*, utilizando también *backtracking*, difiere en que los estados a evaluar dentro de una rama serán elegidos para tomar sólo aquel que resulte como el más prometedor, el que parece llevar de manera idónea a la solución.

2.3.1.1 Algoritmo para backtracking2.

1. Regresa a estado predecesor.
2. No existe estado predecesor termina y regresa valor nulo.
3. No tiene descendientes sin revisar ir a 1.
4. Buscar descendiente más prometedor sin revisar.
5. Termina y regresa estado más prometedor encontrado.

Figura 27. Algoritmo de backtracking2 para la búsqueda hill-climbing.

La *búsqueda hill-climbing* utiliza también *backtracking* como la *búsqueda en profundidad*, pero este *backtracking* difiere del primero (Figura 26) en que no va a regresar el estado alternativo más próximo del estado que fue evaluado, sino que buscará entre los estados alternativos restantes de ese mismo estado al que resulte ser ahora más prometedor, sin importar que se encuentre a su derecha o a su izquierda.

2.3.2 Búsqueda por umbral.

1. Elige los M mejores estados en nivel N.
2. Toma estado de nivel N.
3. Si estado=objetivo entonces termina encontró solución.
4. Genera descendientes de estado en nivel N+1.
5. Si otro estado prometedor en nivel N entonces toma estado, ir a 3.
6. Si número de estados=0 en nivel N termina no encontró solución.
7. Cambia a nivel N+1, N=N+1, ir a 1.

Figura 28. Algoritmo de búsqueda por umbral.

Este *algoritmo* se dirige de nivel en nivel como el de *búsqueda en amplitud*, pero tomando sólo para evaluar en cada nivel, los M estados que parezcan ser más prometedores y no todos los del nivel.

2.3.3 Búsqueda por el mejor nodo.

1. Generar estado.
2. Si estado=objetivo entonces termina encontró solución.
3. Genera descendientes.
4. Revisar todos los estados, estado=estado más prometedor sin revisar ir a 2.
5. No existe estado más prometedor termina sin solución.

Figura 29. Algoritmo de búsqueda por el mejor nodo.

Este *algoritmo* hace la búsqueda guiándose por el mejor estado (no evaluado), encontrado en cualquier parte del árbol de búsqueda, puede encontrarlo en el mismo nivel o rama en que se encuentre actualmente o cambiar a otro del lado derecho o izquierdo, no importa su colocación dentro del *árbol*.

La forma en que los *algoritmos* hacen el recorrido del espacio de búsqueda es lo que los hace diferentes entre sí, marcando el número de estados que se generarán y visitarán, y a partir de este hecho variará su eficiencia.

3. Definición de clases para la programación de búsquedas.

La *clases* que se implementan son dos: la *clase* correspondiente al conjunto de *conocimientos*, que formarán la *base de conocimientos*, y la *clase* que integra el conjunto de *estados de búsqueda* de solución.

La *base de conocimientos* contiene información general del *dominio* del problema a resolver, mientras que los estados toman sólo la información relevante que se obtiene a partir de la *base de conocimientos*.

La *clase* de *base de conocimientos* recibe el nombre de "base_conoc".

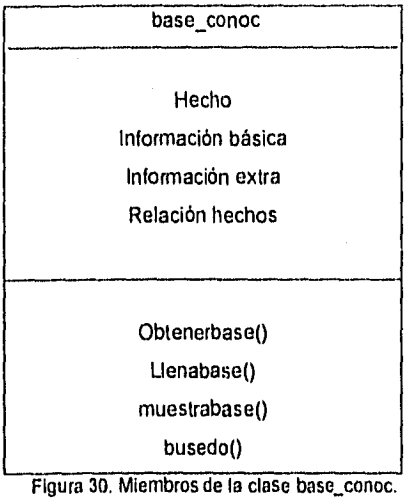


Figura 30. Miembros de la clase base_conoc.

Esta tabla muestra la definición de la *clase* "base_conoc", cuyos atributos son:

hecho = * identificador del *hecho* almacenado*.

inforba = * es la información mínima necesaria que califica al *hecho* y la cual se toma como base para la búsqueda de solución del problema *.

información extra = * si existe más información acerca del *hecho* almacenado que pueda ayudar a encontrar la solución , depende de la aplicación*.

relación hechos = * información que muestre alguna liga o relación entre este *hecho* y los demás, con respecto al problema*.

Los *métodos* que contiene son:

llenabase() = * Este *método* o función ayuda a guardar los *hechos* dentro de la *base de conocimientos*, a partir de la información dada por el usuario *.

obtenerbase() = * Este *método* o función ayuda a obtener los valores de los *hechos* dentro de la *base de conocimientos**.

muestrabase() = * Este *método* o función ayuda a visualizar los *hechos* dentro de la *base de conocimientos* *.

busedo() = * Este *método* permite conocer si el *hecho* está relacionado con el problema a resolver *.

Otra *clase* utilizada es la que representa a cada *hecho* dentro del *espacio de búsqueda* de solución del problema, en el árbol que representa los diferentes recorridos de acuerdo a la búsqueda a realizar, donde cada *nodo* es un estado de búsqueda que corresponde a un *hecho* de la *base de conocimientos*.

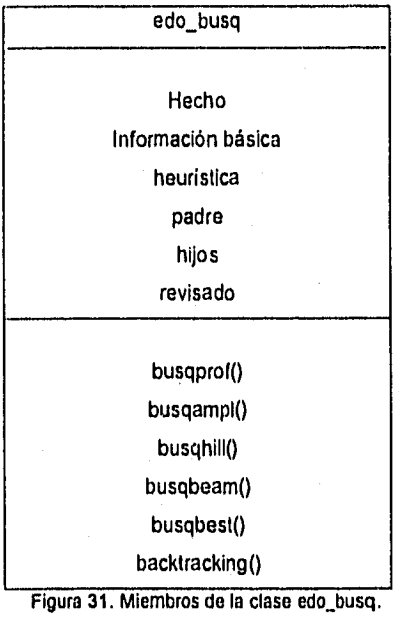


Figura 31. Miembros de la clase edo_busq.

Algunos de los atributos de "edo_busq" de información básica corresponden a los mismos con que cuenta la *clase* de "base_conoc" por lo cual va a heredarlos de la misma, pero además agrega otro que a continuación se mencionan:

padre= * es la liga que lleva hacia el estado anterior, es decir al estado predecesor *.

hijos= * son las ligas que conducen a los estados que surgen del estado presente *.

revisado= * es le información que indica si el estado ha sido ya evaluado o no, durante la búsqueda*.

busqprof()= * es el *método* que se encarga de realizar la búsqueda de solución de acuerdo al *algoritmo* de búsqueda en profundidad*.

busqampl()= * es el *método* que se encarga de realizar la búsqueda de solución de acuerdo al *algoritmo* de búsqueda en amplitud *.

busqhill()= * es el *método* que se encarge de realizar la búsqueda de solución de acuerdo al *algoritmo* de búsqueda escalada de la colina *.

busqbeam()= * es el *método* que se encarga de realizar la búsqueda de solución de acuerdo al *algoritmo* da búsqueda por umbral*.

busqbest()= * es el *método* que sa encarga de realizar la búsqueda de solución de acuerdo al *algoritmo* de búsqueda por el mejor nodo*.

backtracking= * es el *método* que permite hacer el retroceso durante las búsquedas en profundidad y escalada de la colina*.

Como puede notarse la *clase* "edo_busq", tiene atributos y *métodos* que también contiene la *clase* "base_conoc". Esto representa que cada estado de búsqueda está formado por un *hecho* de la *base de conocimientos*, del que hereda esos atributos y *métodos*, por lo tanto es preciso considerar a la *clase* "basa_conoc" como una *clase base* y a la *clase* "edo_busq" como una *clase derivada*, la cual es además una especialización de la *superclase* o *clase base*, ya que añade sus propios *métodos* y atributos para utilizarla de manera específica.

Clase Base:

base_conoc.

Clase Derivada:

edo_busq.

4. Programación de las clases.

```
#include <string.h>
#include <iostream.h>
#include <conio.h>
#include <fstream.h>

#indef Clasbase_h
#define Clasbase_h


class base_conoc{

  public:
        char nombre[20];
        int hauristic;
        int infor;
        int rel1;                    //datos miembro
        int rel2;


        void llenabase();            //métodos miembro
        void despliega();
        void obtenbase();
        base_conoc* busedo(int);

};

#endif
```

Figura 32. Definición del archivo de cabecera Clasbase.h

Dentro de la programación de C++, la *superclase* "base_conoc" se define como lo muestra la Figura 32.

El tipo de correspondiente a los datos miembro aquí definido sólo es como referencia, podría cambiarse por cualquier otro.

La figura siguiente muestra la declaración de la *clase derivada*, estado, de la *superclase* base_conoc, que se encuentra en el archivo de cabecera clasbase.h.

Esta *clase* es guardada en el archivo de cabecera también estado.h.

```
#indef estado_h
#define estado_h
#include <clasbase.h>
#define max 20

class edo_busq : public base_conoc {
  public:
      void edo_busq();
      void edo_busq(edo_busq*,base_conoc);
      edo_busq busqprof(int,edo_busq);
      edo_busq busqampl(int,edo_busq);
      edo_busq busqhill(int,edo_busq);
      edo_busq busqbeam(int,edo_busq);
      edo_busq busqbest(int,edo_busq);
      edo_busq* back(int);
      edo_busq* back(int,int);
      base_conoc busedo(int);
      void imprime();
      ~edo_busq();
protected:
      edo_busq* padre;
      edo_busq* hijos[max];
      int nhijo;
      char rev;
      int hrev;
};

#endif
```

Figura 33. Definición de la clase derivada edo_busq.

Los *métodos* completos de la *clase* edo_busq son mostrados a continuación con sus *métodos* completos:

```
// Programa busqin.h , que define los métodos para realizar las búsquedas inteligentes

#include "estado.h"


edo_busq *provis[max];
edo_busq *revers[max];

//método para localizar los estados descendientes

int edo_busq::busedo(int reg){
int i,x=0;
int bandera=1;
base_conoc *tempo;
 do {
   tempo=new base_conoc;
   tempo=base_conoc::busedo(reg,bandera);
   bandera=0;
   if (tampo){
    provis[x]=new edo_busq(padre,*tempo);
    x++;
   };
 }while (tempo);
return x;
};


//método que realiza la búsqueda en profundidad

edo_busq edo_busq::busqprof(int obj,edo_busq raiz,int num){
edo_busq *revis[max],superior,inferior;
int i=0,j,t,dif,r=0,z=0,x=0;;
revis[i]=new edo_busq;
 revis[i]=&raiz;
 while(revis[i]->infori=obj && i<num){
dif=obj-revis[i]->infor;
    if (dif<0){
        if(obj-superior.infor<dif || obj-superior.infor==obj)
          superior=*revis[i];  //superior al pedido
        }
        else
          if(obj-inferior.infor>dif)
            inferior=*revis[i];       //inferior inmediato al pedido
    z=0;x=0;
```

```
      z=edo_busq::busedo(revis[i]->rel2);
      revis[i]->nhijo=z;
      if(z>0) {
            for(j=0;j<z;j++)
              revis[i]->hijos[j]=provis[j];
            revers[r]=new edo_busq;
            revers[r]=revis[i];
            r++;
            if(revis[i]->nhijo>revis[i]->hrev)[
              i++;
              revis[i]=new edo_busq;
              revis[i]=revis[i-1]->hijos[revis[i-1]->hrev];
              revis[i-1]->hrev++;
            };
      } //if
      else{
            i++;
            revis[i]=revis[i]->back(r-1);
      };
  }; //while()
  if (revis[i]->infor!=obj){
    moveto(150,450);
    outtext("No se encontró la cantidad buscada, las más cercanas son: ");
    moveto(470,350);
    muestra(superior.infor);
    moveto(470,400);
    muestra(inferior.infor);
    return superior;
  }//if
  moveto(470,10);
  outtext("Cantidad encontrada: ");
  moveto(425,110+10*(busqpr+1));        //edo_busq revisado
  muestra (revis[i]->infor);

  return *revis[i];
};


//método para el backtracking de la búsqueda en profundidad

edo_busq* edo_busq::back(int atras){
edo_busq *s; //circulo na;
int encu=0;
  while(!encu && atras>=0){
    if (revers[atras]->hrev<revers[atras]->nhijo){
          s=revers[atras]->hijos[revers[atras]->hrev];
          encu=1;
          revers[atras]->hrev++;  //1
    }
    else
          atras--;
  }; //while
  if (atras<0)
    s=NULL;
```

```
 return s;
}

// método para la búsqueda en amplitud.

 edo_busq edo_busq::busqampl(int obj,edo_busq raiz,int num){
 edo_busq *nivel1[max],*nivel2[max],superior,inferior;
 int i,j,t,m,con=0,dif,desc;
 i=0; m=0;
 nivel1[i]=new edo_busq;
 nivel1[i]=&raiz;
 while(nivel1[i]->infor!=obj && busqam<num+1){  //para cuando el padre sea null
      dif=obj-nivel1[i]->infor;
     if (dif<0){
          if(obj-superior.infor<dif || obj-superior.infor==obj)
            superior=*nivel1[i];//superior al pedido
     }
     else
          if(obj-inferior.infor>dif)
            inferior=*nivel1[i];       //inferior inmediato al pedido
          t=nivel1[i]->busedo(nivel1[i]->rel2);
          nivel1[i]->nhijo=t;
          if(t>0) {
           for(j=0;j<t;j++){
             nivel1[i]->hijos[j]=provis[j];
             nivel2[m]=provis[j];
             m++;
           };
          };
          if(i<con-1){
           i++;
          }
          else
           if(m>0){
             j=0;
             while(nivel1[j]->nhijo==0)
               j++;
             for (i=0;i<m;i++)
                   nivel1[i]=nivel2[i];
             con=i;
             m=0;i=0;
           };
}; //while()
 f(nivel1[i]->infor!=obj){
   moveto(470,10);
   outtext("No se encontro la cantidad buscada, las m s cercanas son: ");
   moveto(470,350);
   muestra(superior.infor);
   moveto(470,400);
   muestra(inferior.infor);
   return superior;
 }
 return *nivel1[i];
```

```
};

//método que realiza la búsqueda escalada de la colina

edo_busq edo_busq::busqhill(int obj,edo_busq raiz,int num){
edo_busq *prometedor[max],*temp,*temp2,superior,inferior;
int i=0,j,t,dif,r=0;
prometedor[i]=new edo_busq;
  prometedor[i]=&raiz;
while(prometedor[i]->infor!=obj && i<num){
        dif=obj-prometedor[i]->infor;
        if (dif<0){
          if(obj-superior.infor<dif || obj-superior.infor==obj)
            superior=*prometedor[i];//superior al pedido
        }
        else
          if(obj-inferior.infor>dif)
            inferior=*prometedor[i];      //inferior inmediato al pedido
        t=inferior.busedo(prometedor[i]->rel2);
        prometedor[i]->nhijo=t;
        prometedor[i]->rev='s';
        if(t>0) {
          for(j=0;j<t;j++){
            if(t>0){
             revers[r]=new edo_busq;
             revers[r]=prometedor[i];
             r++;
            }
          j=0;
          temp2=prometedor[i]->hijos[j];
          for(j=0;j<t-1;j++){
           temp=prometedor[i]->hijos[j+1];
           if(temp2->heuristic<temp->heuristic && temp->rev=='n'){
            temp2=temp;
          };//if
          };//for
          prometedor[i]->hrev++;
          i++;
          prometedor[i]=NULL;
          prometedor[i]=temp2;
    }//if
    else{
          i++;
          prometedor[i]=new edo_busq;
          prometedor[i]=prometedor[i]->back(r-1,prometedor[i-1]->heuristic);
    };
}; //while()
if(prometedor[i]->infor!=obj){
   moveto(470,10);
   outtext("No se encontro la cantidad buscada, las m s cercanas son: ");
   moveto(470,350);
   muestra(superior.infor);
   moveto(470,400);
```

```
   muestra(inferior.infor);
   return superior;
  }
  return *prometedor[i];
};

//método que realiza el backtracking para la búsqueda escalada de la colina

edo_busq* edo_busq::back(int atras,int heurist){
edo_busq *s,*temp,*temp2;
int encu=0,x,y,dif;
  while(!encu && atras>=0){
if (revers[atras]->hrev<revers[atras]->nhijo){
      revers[atras]->hrev++;
      x=0;temp=revers[atras]->hijos[x];
      while(x<revers[atras]->nhijo && temp->rev=='s'){
            temp=revers[atras]->hijos[x];
            x++;
      };
      if(revers[atras]->nhijo==revers[atras]->hrev)
           encu=1;
      for (x=0;x<revers[atras]->nhijo;x++){
           dif=heurist-temp->heuristic;
           temp2=revers[atres]->hijos[x];
           if (dif>heurist-temp2->heuristic && temp2->rev=='n')
            temp=temp2;  //=
      } //for
      temp->rev='s';
      encu=1;
    }//if
    else
      atras--;
};
  if(atras<0)
    return NULL;
  return temp;
};

//método que realiza la búsqueda por umbral

edo_busq edo_busq::busqbeam(int obj,edo_busq raiz){
edo_busq *nivel2[max/2],*nivel1[max],*temp,temp2,inferior,superior;
int l,j,t,u,n,con,dif;
   l=j=t=n=con=dif=0;
while(nivel1[i]->infor!=obj && con>=0){  //para cuando el padre sea null
dif=obj-nivel1[i]->infor;
          if (dif<0){
            if(obj-superior.infor<dif || obj-superior.infor==obj)
            superior=*nivel1[i];//superior al pedido
           }
```

```
        else
          if(obj-inferior.infor>dif)
          inferior=*nivel1[i];       //inferior inmediato al pedido
        t=nivel1[i]->busedo(nivel1[i]->rel2);  //debe inicializar a cont de la busq en 0
        if(t>0)
          for(j=0;j<t;j++){
            nivel1[i]->hijos[j]=provis[j];
          }//for
        n=0;
        while(n<(t+1)/2){
          j=0;
          temp=provis[j];
          while(j<t-1){
            if (provis[j]->heuristic<provis[j+1]->heuristic && provis[j+1]->rev=='n')
                temp=provis[j+1];
            j++;
          };
          temp2.rev='s';
          nivel2[n]=temp;
          nivel2[n]->rev='s';
          n++;
        };
        if(i<con)
          i++;
        else{
          for (i=0;i<n;i++)
            nivel1[i]=nivel2[i];
            con=i-1;
            i=n=0;
        };
}; //while()
if(nivel1[i]->Infor!=obj){
   moveto(470,10);
   outtext("No se encontro la canlidad buscada, las m s cercanas son: ");
   moveto(470,350);
   muestra(superior.infor);
   moveto(470,400);
   muestra(inferior.infor);
   return superior;
 }
 return *nivel1[i];
};


//método que realiza la búsqueda por el mejor nodo

edo_busq edo_busq::busqbest(int obj,edo_busq raiz,int num){
edo_busq *mejor,*revis[max],*probable[max],*mejor2,inferior,superior;
int i,j,t,dif,l=0;
 i=j=t=dif=0;
 revis[i]=new edo_busq;
 revis[i]=&raiz;
while(revis[i]->infor!=obj && i<num){  //para cuando el padra sea null
```

```
dif=obj->revis[i]->infor;
   if (dif<0){
     if(obj->superior.infor<dif || obj->superior.infor==obj)
          superior=*revis[i];//superior al pedido
   }
   else
     if(obj->inferior.infor>dif)
          inferior=*revis[i];       //inferior inmediato al pedido
     revis[i]->rev='s';
     t=revis[i]->busedo(revis[i]->rel2);
     revis[i]->nhijo=t;
     l=0;
     for(j=0;j<t;j++)
          revis[i]->hijos[j]=provis[j];
     j=0; l=0;
     while(j<=i){
          if(revis[j]->rev=='n'){
            probable[l]=revis[j];
            l++;
          }
          else
            for (int s=0;s<revis[j]->nhijo;s++)
              if(revis[j]->hijos[s]->rev=='n'){//.heuristic<probable[s]->heuristic)
                probable[l]=revis[j]->hijos[s];
                l++;
              };
            j++;
     };//while
     if (l>0){
          mejor=probable[0];
          for (int s=1;s<l;s++)
           if(mejor->heuristic<probable[s]->heuristic)
            mejor=probable[s];
     }
     i++;
     revis[i]=mejor;
     revis[i]->rev='s';
     busqme++;
  };//while
if(revis[i]->infor!=obj){
    moveto(470,10);
    outtext("No se encontro la cantidad buscada, las m s cercanas son: ");
    moveto(470,350);
    muestra(superior.infor);
    moveto(470,400);
    muestra(inferior.infor);
    return superior;
  }
  return *revis[i];
};
```

La *clase base* "base_conoc" define los datos básicos que se requieren para definir una *base de conocimientos* estándar e iniciar el desarrollo de un Sistema Inteligente.

Los *métodos* contenidos por esta *clase* son igualmente los considerados como esenciales para la creación, visualización y manipulación de la *base de conocimientos*. Con esto se indica que la *base de conocimientos* puede tener más datos y *métodos* miembros de los que aquí se indica, pero inicialmente debe esperarse que tenga los ya indicados.

Esta *clase* se toma principalmente como una *clase base*, precisamente por lo dicho anteriormente, es decir, que se piense en dar más información a la *base de conocimientos*, creando a partir de la *clase* "base_conoc" otra que al integrar más datos y *métodos* miembros de acuerdo con su aplicación sea una *clase derivada* especializada de esta primera *clase*.

La *clase* "edo_busq" es la encargada de definir los datos y *métodos* básicos también requeridos para la realización de Sistemas Inteligentes, a través de sus miembros se llevan acabo las búsqueda inteligentes, las que son una parte primordial en los Sistemas Inteligentes.

Como ya se mencionó en párrafos anteriores debido a que los *estados de búsqueda* se refieren al conjunto de *hechos* que se tienen disponibles en la *base de conocimientos* para realizar la búsqueda, se creó como una *clase derivada* de la *clase* "base_conoc", de la cual hereda sus datos miembro, es decir, nombre, heuristic, infor y rel1 y rel2.

Los *métodos* miembro no son tomados por *herencia* y más aún uno de éstos es redeclarado en esta *clase derivada*, sin embargo, son completamente distintos estos *métodos*, tanto por sus acciones como por sus argumentos y valores de retorno, podía haberse llamado de distinta forma en alguna de las dos *clases* pero, para denotar que se utilizan ambos dentro de la localización de *hechos* en la *base de conocimientos*, fueron denominados con el mismo nombre.

Los *métodos* referidos en el párrafo anterior son:

base_conoc* busedo(int,int); y
int busedo(int);

El primero, está definido dentro de la *clase* "base_conoc" y es utilizado para localizar dentro de la *base de conocimientos* los *hechos* que vayan a ser manipulados por las búsquedas de solución debido a su relación con los demás *estados de búsqueda*, es por eso que el valor que retorna después de su ejecución es del tipo de *base de conocimientos*, es decir, de tipo "base_conoc".

El segundo *método*, se encuentra dentro de la *clase* "edo_busq" y retorna el número entero correspondiente a la cantidad de *estados de búsqueda* a crear a partir de los *hechos* encontrados que tengan relación con el *nodo* o estado actualmente verificado, para lo que realiza llamados al *método* base_conoc* busedo(int,int).

Todo esto significa que, el *método* busedo contenido en la *clase* "edo_busq" hace uso del *método* busedo pero propio de la *clase* "base_conoc" para la obtención más *estados de búsqueda* para la solución del problema, esta es la razón por la que tienen el mismo nombre.

La *clase* "edo_busq" muestra el *polimarfismo* en los *métodos*:
edo_busq(), edo_busq(edo_busq*,base_conoc), edo_busq* back(int) y edo_busq* back(int,int).

En esos *métodos* puede observarse claramente la reutilización del mismo nombre y realizan funciones diferentes, para dar a conocer al compilador cuál *método* es el que se utilizará se diferencian por su número de parámetros.

En el *método* edo_busq* back(int), sólo se pide un parámetro entero, mientras que en el otro edo_busq* back(int,int) son dos los que se piden, al invocar a cualquiera de los dos se le da el o los parámetros debidos y con ello se da una diferencia muy significativa para lograr diferenciar entre las llamadas que se hacen a uno u otro.

El *método* edo_busq* back(int), se utiliza para hacer un *backtracking* dentro del *algoritmo* de *búsqueda ciega, primero en profundidad,* el otro base_conoc* busedo(int,int); realiza el *backtracking* empleado en la *búsqueda heurística hill-climbing.*

En cuanto al *polimorfismo* de constructores, edo_busq() y edo_busq (edo_busq*, base_conoc) se requiere para la construcción de objetos de la *clase* "edo_busq", donde el primero se crea con la inicialización de sus datos miembros y el segundo para crear un objeto "edo_busq" a partir del conocimiento obtenido de la *base de conocimientos.*

5. Muestra del uso de las directivas.

Para explicar mejor el uso de los *algoritmos* a través de estas *clase* definidas, a continuación se muestra este caso contable-administrativo, el cual está hecho con datos ficticios que sólo pretenden ejemplificar el uso de las librerías y no realizar por completo un Sistema Inteligente.

Suponiendo que los accionistas de la empresa Elecktra, S.A. requieren conocer continuamente la situación en que están sus sucursales, desean ellos conocer qué sucursal ha tenido cierto número de ventas para apoyar con ello la toma de decisiones que sobre sus sucursales se tome.

Para este motivo se ha creado una *base de conocimientos* que contiene *hechos* referentes a sus sucursales con datos a partir de los cuales se podrá evaluar a cada una y crear los *estados de búsqueda* de solución a problemas, que en este caso consiste en encontrar cierta cantidad de ventas objetivo.

De acuerdo con las *clases* definidas (Figuras 32 y 33), anteriormente, los *hechos* de la *base de conocimientos* se referirán a cada una de las tiendas Elecktra consideradas, la información básica es la cantidad vendida por cada tienda en el último semestre, las relaciones serán de acuerdo a las zonas físicas en que se encuentran cada tienda y la información heurística será obtenida a partir del número de habitantes de la región, el número de tiendas de otras empresas que signifiquen competencia y un calificativo dado a la región en que se encuentre la tienda.

Para desarrollar este caso se diseño una *clase* para los conocimientos específicos acerca del problema, la cual es una especialización de la *clase* "base_conoc", es decir, que hereda de ésta sus datos y *métodos* miembros y además especifica otros propios de la aplicación que tendrá esta nueva *clase derivada*.

La *clase* recibe el nombre de "base_empresa" y se define como a continuación se indica:

```
class base_empresa:public base_conoc{
  public:
        void llenabase();
        void despliega();
        int obtenbase();
        int obtenbasheur();
        int metheurist();
  private:
        int habit;
        int competencia;
        char indice;
};
```

Figura 34. Declaración de la clase base_empresa.

Las funciones de llenabase(), obtenbase() y despliega() son redeclaradas para hacerlas acordes con las necesidades de la nueva *clase* que también añade los datos de habit, competencia y índice.

habit=" número de habitantes de la región en que se encuentra la tienda".

competencia=" número de tiendas del mismo ramo que están en esa misma región".

indice=" refleja un índice cualitativo obtenido para esa región de acuerdo con los estudios que de ésta se hayan hecho previamente".

Para la *base de conocimientos* serán utilizados archivos de datos para almacenar y recuperar la información acerca de las tiendas.

La función obtenbaseheur() difiere de la de obtenbase() en que además de obtener la información básica de la base va a hacer la evaluación a través de la función metehurist() para manejar la información heurística. Por esto para hacer las *búsquedas heurísticas* se va emplear esta función para obtener la *base de conocimientos*, mientras que para las *búsquedas ciegas* bastará con el uso de la función obtenbase().

Para la evaluación de la información que integra la *base de conocimientos* de la empresa, se crean instancias de la *clase* "edo_busq", con lo que se manejarán las búsquedas.

Debido al objetivo de la investigación que es educativo del trabajo las búsquedas son mostradas de forma visual, es decir, que puede verse qué *hechos* va recorriendo, cuáles son creados y cuáles son recorridos, indicando el número total de *nodos* recorridos o evaluados y de los creados en cada uno de los tipos de búsqueda, con el objetivo de verificar el rendimiento de cada uno de los tipos de búsquedas inteligentes.

El producto para la demostración del uso de las directivas está integrado por los siguientes módulos:

Base de conocimientos.
Búsquedas.
Explicación.
Salida.

Dentro del módulo de Base de conocimientos se pueden almacenar *hechos* referentes al tipo de *base de conocimientos* empleado, así como ver todos los *hechos* almacenados en la *base de conocimientos.*

El módulo de Búsquedas como su nombre lo indica es el que lleva a la utilización de las búsquedas inteligentes sobre la *base de conocimientos* obtenida.

Las búsquedas están divididas en sus dos categorías: *búsquedas ciegas* y búsquedas heurísticas, para dar finalmente datos estadísticos del número de *nodos* creados y evaluados en cada una de las variantes de estas búsquedas y conocer su rendimiento.

La explicación muestra aspectos de la programación de las *clases* básicas, es decir, de las *clases* que corresponden a las cabeceras, "base_conoc" y "edo_busq". También hace referencia a cada una de las búsquedas inteligentes, dando sus principales características y mostrando en forma gráfica su operación. Esta gráfica se refiere a la creación del *árbol* que contiene *estados de búsqueda* permitiendo ver el recorrido que a través de tales *estados* se hacen en las búsquedas inteligentes.

5.1 Especificaciones para el usuario.

Nombre.
Demcin (demostrador para C++ inteligente)

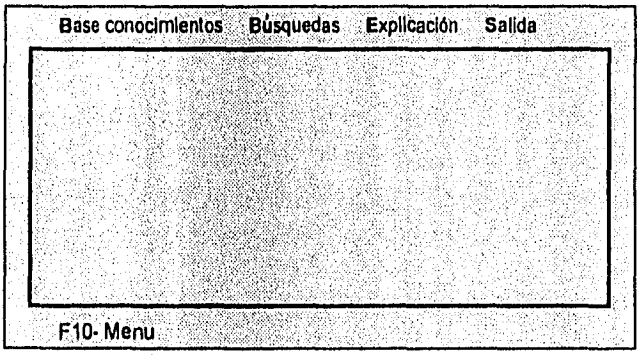
Objetivo.
Ejemplificar para el usuario la funcionalidad de los archivos de cabecera (#includes) creados para llegar a un C++ Inteligente y a través de los cuales puede iniciarse en el desarrollo de **Sistemas Inteligentes**, así como mostrarle las características de las búsquedas inteligentes, que son elementos esenciales para el desarrollo de **Sistemas Inteligentes**.

Para la utilización del demostrador se requiere de un equipo PC 386 en adelante con monitor a color para la adecuada visualización de los gráficos.

Para iniciar su uso teclear estando el programa en la unidad de disco A:

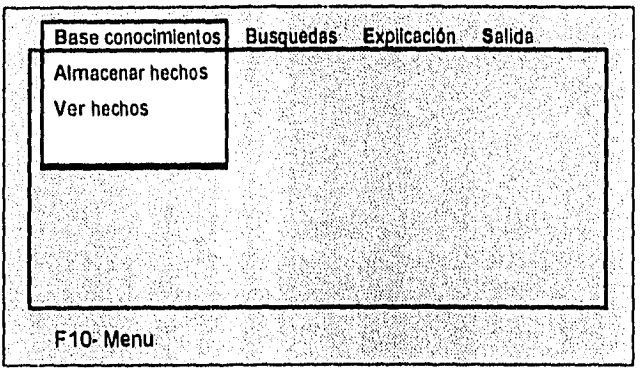
A:\> **demcin** <ENTER>

A continuación de la pantalla de presentación se presenta el menú:



Base conocimientos.

Es la opción que se encarga de manipular directamente la *base de conocimientos* almacenada, permite como el submenú muestra:



Almacenar hechos.

En esta pantalla se introducen los datos referentes a cada *hecho*, los que son:

Sucursal. Identificador de la sucursal a almacenar. Es de máximo 20 caracteres.

Ganancia. Número entero que indica la ganancia obtenida por la sucursal. Se expresa en millones, p.e., si se trata de 123 millones sólo introducir el número 123.

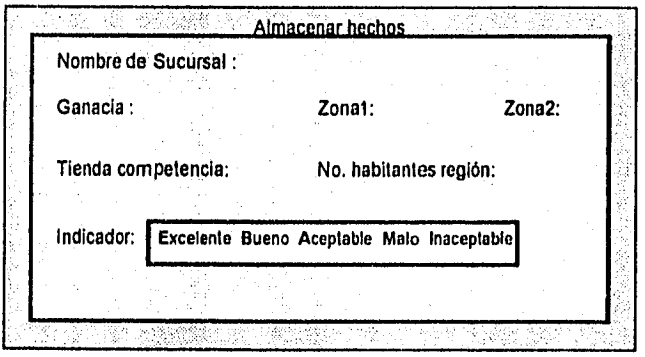
Zona1 y Zona2. Representan un número identificador dado a cada sucursal para clasificarlas: va desde el número 1 en adelante, con éstos se indican las relaciones entre las sucursales.

Número de habitantes. Corresponda al número de habitantes (en miles), que tiene la región en que se encuentra la sucursal.

Competencia. Es el número de empresas de la competencia que se encuentran en la misma región que la sucursal.

Indicador. Calificativo obtenido para la región en que se encuentra la sucursal con respecto a la situación de la población, servicios públicos, condiciones generales favorables o no de la región, se obtiene por los estudios de mercado hechos. Es de acuerdo con la siguiente tabla:
E- Excelente.
B- Bueno.
A- Aceptable.
M- Malo.
I- Inaceptable.

La información aquí obtenida se guarda en un archivo en disco para su posterior utilización.



Ver hechos.

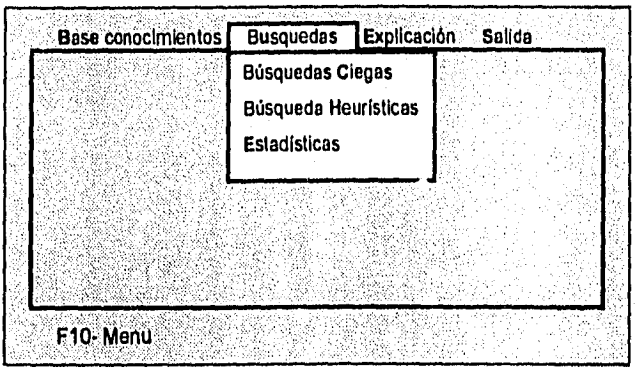
Esta pantalla muestra todos los *hechos* contenidos en la *base de conocimientos.*

Búsquedas.

En esta opción pueden realizarse los 5 tipos de búsquedas inteligentes incluidos, estén agrupados en *búsquedas ciegas* y *búsquedas heurísticas*. Pudiendo también consultarse aspectos acerca del rendimiento de cada tipo de búsqueda los que se indican por el número de *nodos* creados y los evaluados durante la búsqueda.

Para evaluar el rendimiento de las búsquedas debe de ser sobre el mismo *hecho* a buscar, por lo que primero se introduce el valor objetivo a buscar por los tipos de búsqueda.

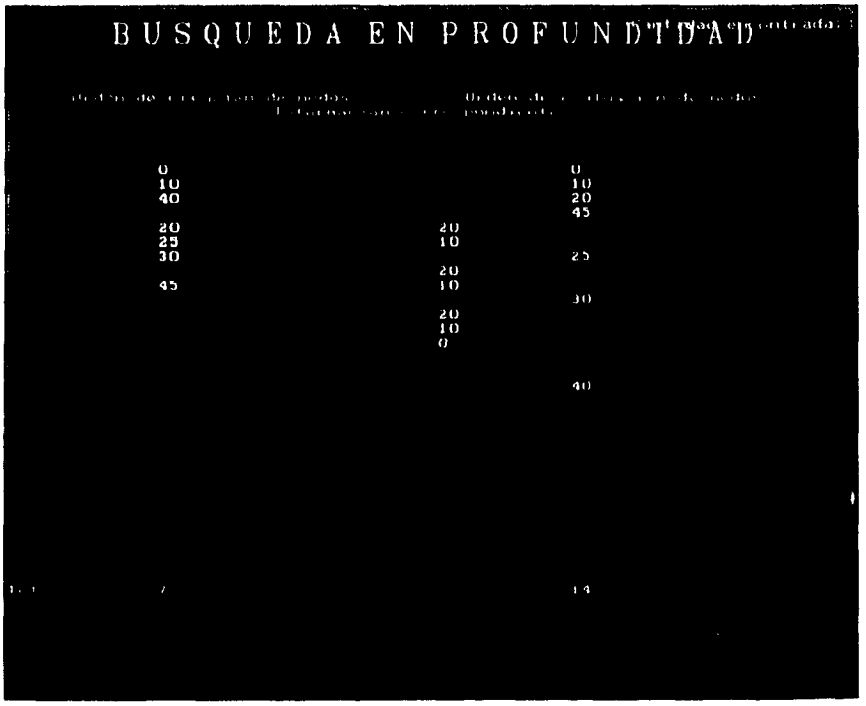
Posteriormente se elige entre:



La opción de Búsquedas Ciegas permite elegir entre sus dos tipos.
Búsqueda en Profundidad.
Búsqueda en Amplitud.

La opción de Búsqueda Heurísticas entre:
Búsqueda de Escalada de la colina.
Búsqueda por Umbral.
Búsqueda por el Mejor nodo.

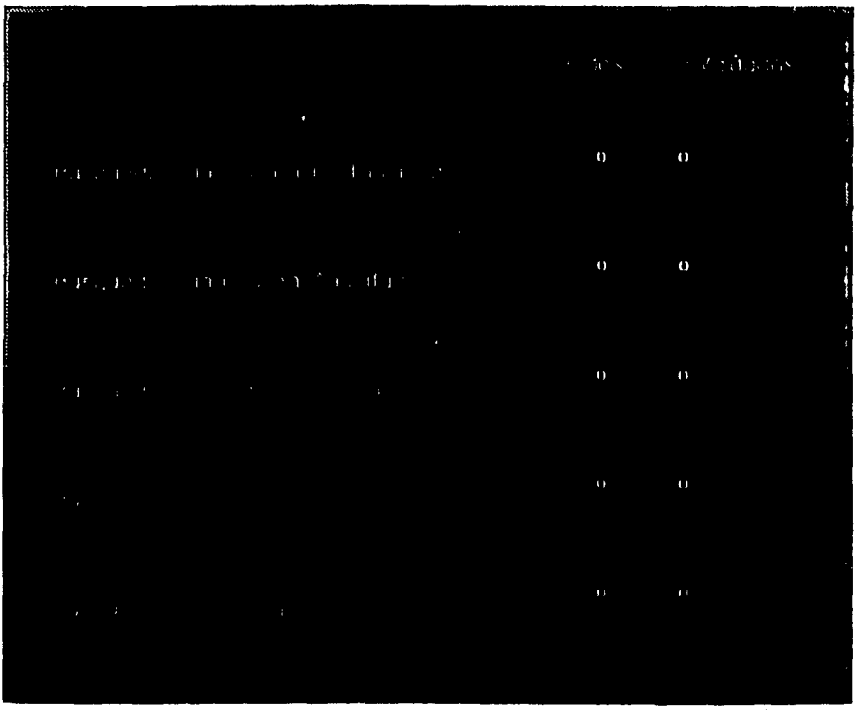
En cada una de los tipos de búsqueda se muestra en pantalla información acerca de cada sucursal que va siendo creada y evaluada durante la búsqueda, al final de la búsqueda se indica el total de los *nodos* creados y evaluados. Si la cantidad objetivo se encontró se muestra la información de la sucursal perteneciente, sino se encontró se dan los valores correspondientes a la sucursal que tenga el valor inferior y la que tiene el valor superior inmediatos.



Estadisticas.

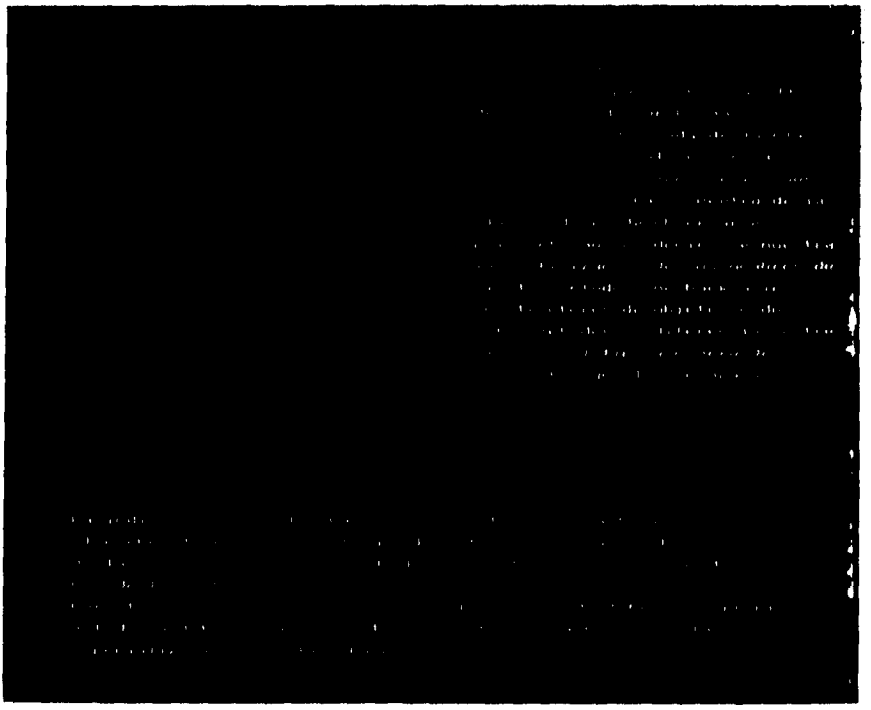
Se presenta el resultado total de los 5 tipos de búsquedas realizados para la localización de un mismo objetivo, por lo que es recomendable utilizarla de manera posterior a la elección de los 5 tipos de búsquedas.

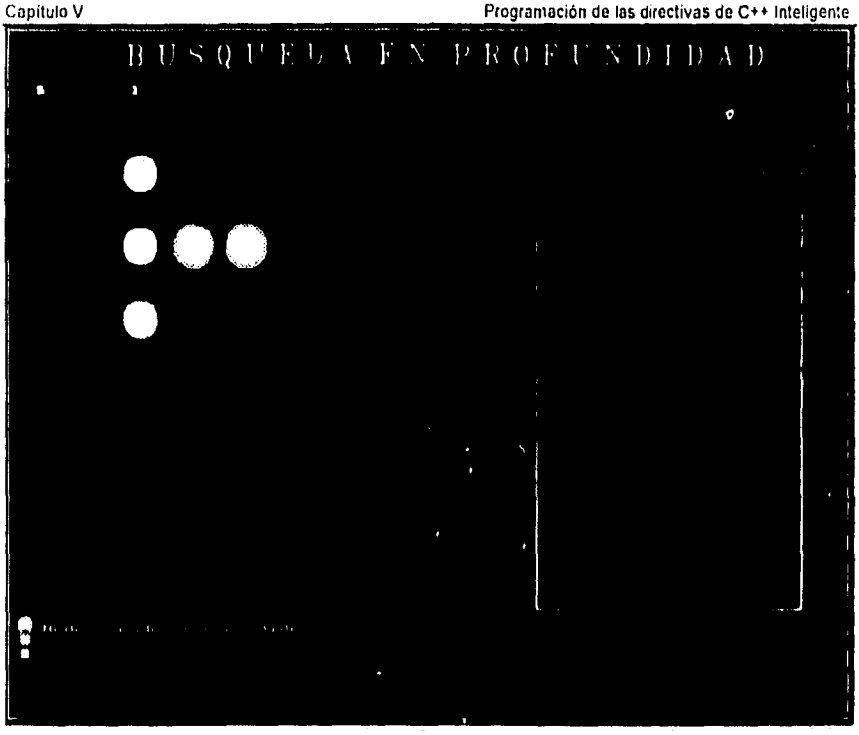
Permite hacer una comparación entre el rendimiento de las búsquedas de acuerdo con el número total de *nodos* creados y evaluados en cada una.



Explicación.

Ayuda a la comprensión de desarrollo de las búsquedas inteligentes por medio de una explicación sintetizada de las *clases* programadas, de las caracteristicas de cada tipo de búsqueda y de una ejemplificación gráfica por medio de la representación de un árbol para la creación y recorrido





Salida.

Para regresar al símbolo del sistema y salir de la ejecución del demostrador Demcin.

Conclusiones.

El resultado de los desarrollos de **Inteligencia Artificial** ha permitido establecer que encuentra sus principales logros en las áreas de *robótica* y *sistemas expertos*, en las que ya cuenta con aplicaciones totalmente confiables utilizadas en los sectores de servicios e industrial mientras que sus principales limitaciones están en la interpretación del *lenguaje natural*, en la que requiere considerar aspectos de sentido común que el humano utiliza y que no han podido representarse en la máquina, sin embargo, ha contribuido mucho al mejoramiento de las ciencias de la computación en beneficio del propio ser humano y a nuevos descubrimientos acerca de la inteligencia humana.

En general, la **Inteligencia Artificial** ha encontrado muchos obstáculos para sus desarrollos, ya que desde sus orígenes se ha esperado mucho más de sus aplicaciones de lo que realmente se ha logrado, por lo que no es fácil encontrar empresas que confíen y se interesen en hacer inversiones para estos desarrollos lo que contribuye a que se encuentre aún en fase de investigación sin poder comercializar con sus productos, en la mayoría de los casos.

En lo que respecta a los **Sistemas Inteligentes**, no han tenido aún un importante despegue en cuanto a su desarrollo, ya que por un lado algunos autores los consideran parte integral de la **Inteligencia Artificial** lo que hace encontrar los obstáculos que la acompañan como se describió anteriormente, y por otro lado no existen herramientas o lenguajes que sean accesibles para todos los desarrolladores de sistemas con los que pueda facilitarse la programación de los **Sistemas Inteligentes**, estos aspectos hacen ver poco atractiva su realización.

Los **Sistemas Inteligentes** en lugar de formar parte de la **Inteligencia Artificial** son en realidad sistemas independientes que utilizan o se basan en elementos de ésta, pero que no surgen de la misma.

Los **Sistemas Inteligentes** son de gran utilidad en el apoyo al personal encargado de la toma de decisiones, es decir, en la emisión de juicios acerca de situaciones particulares, ya que contienen hechos que pueden ser cualitativos y cuantitativos referentes a un dominio específico, los que pueden llegar a cambiar con frecuencia, teniéndose que anticipar a circunstancias, considerando la incertidumbre y contando en todo momento con la experiencia y conocimientos de personas expertas en la toma de decisiones.

También contribuyen al incremento de la productividad, porque los **Sistemas Inteligentes** desarrollados hacen sus tareas específicas en menos tiempo que un trabajador por no sufrir interrupciones constantes; ayuda al mejoramiento de la calidad con la reducción del número de errores y al logro de mayor precisión.

Con la ampliación del desarrollo de los **Sistemas Inteligentes** podrá ser mayor el número de organizaciones e individuos que se beneficien con su aplicación. Con los **Sistemas Inteligentes,** como todos los desarrollos de la computación creados son para aumentar el potencial de la actividad humana y no para sustituirlo.

La **Inteligencia Artificial** trata de representar información lo más semejante posible a como la entendemos los humanos en nuestros propios procesos mentales, por lo que se crearon diversos esquemas que pudieran soportar información expresada como *símbolos*, las *representaciones del conocimiento*, y también métodos de búsqueda para su manejo, búsquedas inteligentes y éstas son las partes principales que para el desarrollo de los **Sistemas Inteligentes** se consideran con respecto a la **Inteligencia Artificial**.

Los **Sistemas Inteligentes** son grandes y complejos por lo que las ventajas de la **Programación Orientada a Objetos** se presentan como de gran utilidad para su programación.

De los lenguajes empleados para **Programación Orientada a Objetos** el más utilizado para realizar aplicaciones es C++, debido a que tiene sus bases en uno de los lenguajes de mayor difusión internacional, el lenguaje C, aunque por este motivo no es un lenguaje puro de **Programación Orientada a Objetos** como Smalltalk e Eiffel, sin embargo, permite desarrollar las características distintivas de la Programación Orientada a Objetos así que se considera como uno más de sus lenguajes.

La definición de *clases* en base al lenguaje C++, permite una *representación del conocimiento* mixta, al incluir tanto los *hechos* como los *métodos* que harán uso de ellos, es decir, el qué y cómo se utilizará el *conocimiento*.

Los *objetos*, como Feigenbaum lo indica, son uno de los tipos de *conocimiento* existente, a través de los cuales se representan aspectos tangibles e intangibles de la realidad, por lo que la **Programación Orientada a Objetos** sí permite una forma válida de *representación del conocimiento* de un **Sistema Inteligente**.

Con la programación de los archivos de cabecera (#include) pueden adaptarse las búsquedas inteligentes mediante una visión del lenguaje de C++ llevándolo hacia un lenguaje inteligente, permitiendo la creación de un mayor número de aplicaciones, ya que hace posible que más personas con mayor facilidad puedan hacer la programación de los **Sistemas Inteligentes**.

La elección de las búsquedas incluidas en las directivas será de acuerdo al tipo de problema que se desee resolver, no puede generalizarse una como la más adecuada, puede decirse simplemente que es recomendable utilizar un tipo de *búsqueda ciega* cuando se desee la solución precisa, sin importar los recursos que puedan utilizarse para lograrla, o en casos en que no se cuente con suficiente información como para hacer una evaluación heurística. Mientras que la *búsqueda heurística* es considerada como la

más recomendable en los casos en que el número de casos a evaluar sean muchos, por lo que revisar a todos llevaría mucho tiempo y altos costos, cuando se tenga la suficiente información como para poder aplicar un criterio o evaluación heurístico y cuando no se busque una respuesta plenamente precisa, sino simplemente satisfactoria.

De acuerdo con la demostración incluide acerca del funcionamiento de las búsquedas se distingue la operación de las *búsquedas heurísticas* y de las *búsquedas ciegas* obteniendo además los datos estadísticos que permiten valorar en que casos puede ser más recomendable cada una de estas dos categorías de búsquedas.

La programación de **Sistemas Inteligentes** se facilita con el lenguaje C++ Inteligente debido a que se basa en aspectos más sencillos para su comprensión y utilización, así las personas que conocen la **Programación Orientada a Objetos** con C++, principalmente, encuentran una nueva vertiente para sus desarrollos pudiendo ahora realizar **Sistemas Inteligentes**, logrando también un incremento importante en el desarrollo de **Sistemas Inteligentes**.

En México existe un gran número de personas del área de *informática* que conocen y utilizan la **Programación Orientada a Objetos** e los que podría introducirse el desarrollo de **Sistemas Inteligentes** y también al conocimiento de los elementos de la **Inteligencia Artificial** así como a estudiantes que cuenten con alguna materia relacionada con la **Inteligencia Artificial** quienes pueden encontrar una forma más fácil de comprenderla.

Indice de figuras.

Indice de cuadros.

Glosario.

Abstracción. Es la separación de los detalles de un objeto, realizada por el diseñador de software, enfocándose en lo que es el objeto.

Álgebra booleana. Proceso de razonamiento o sistema deductivo utilizando la lógica simbólica.

Algoritmo. Conjunto de pasos que siguen una secuencia lógica para la solución general de problemas, especificando su principio y final.

Algoritmos genéticos. Técnicas de búsqueda derivada de los principios de la genética natural, es útil donde se encuentran un gran número de hipótesis, permite aprender.

Árbol. Estructura de datos compuesta de nodos que son ligados de una forma jerárquica. La parte más alta se conoce como raíz, la cual puede tener diversos nodos hijos a los que se liga, los nodos hijos pueden o no tener a su vez otros hijos, pero cada uno tiene un solo padre.

Arquitectura RISC. Reduced Instruction Set Computer. Tecnología de computadora basada en un procesedor diseñado para ejecutar un pequeño número de instrucciones sencillas a gran velocidad, emplea el proceso de encauzamiento (pipeling) y la memoria temporal.

Atributo. Las características y propiedades de un objeto, a través de las que se define.

Autómata. Sistemas que pueden adoptar diversos estados discretos durante un tiempo y en función de los estados adoptados anteriormente y la entrada presentada.

Backtracking. Estrategia de control utilizada en la IA en el proceso de búsqueda, por el cual puede retrocederse a partir de un resultado incorrecto hasta el punto en que se considera estaba correcto.

Backward chaining. Ver encadenamiento regresivo o hacia atrás.

Base de conocimientos. La base de conocimientos de manera simbólica representa el conocimiento humano acerca de hechos e información general requeridos para un sistema, así como heurísticos, tales como juicios, intuición y experiencias sobre determinada área del saber y procedimientos necesarios para solucionar un problema.

Best-first. Búsqueda heurística que da comienzo en el mejor valor encontrado, o lugar más óptimo dentro del árbol de búsqueda.

Bottom-up. Diseño en el que un programa se hace completando un paso antes de continuar con el otro.

Búsqueda ciega. Tipo de búsqueda contraria a la heuristica, en la que se hace una revisión exhaustiva entre las diferentes alternativas durante la búsqueda de la solución de un problema, por lo que origina un gran consumo de tiempo y recursos, pero asegura encontrar la solución, si ésta existe.

Búsqueda en amplitud. Búsqueda ciega en la que se evalúa cada alternativa posible dentro de los estados de búsqueda, haciendo el recorrido de nivel en nivel, es decir, de forma horizontal.

Búsqueda en profundidad. Tipo de búsqueda ciega, donde se explorarán todas las alternativas posibles encontradas en el árbol de manera vertical, va de rama en rama.

Búsqueda heurística. Método de búsqueda de la solución de un problema en el que se utiliza información relevante acerca del problema, buscando sólo en los estados que aparecen como más prometedores, ahorrando tiempo y recursos, sin embargo, no garantiza el encontrar la solución más óptima del todo.

Cálculo del predicado. Forma de representación del conocimiento en forma de proposiciones, basado en la lógica formal.

Clase. Estructura jerárquica de cosas según la POO. Las clases y subclases son conjuntos de una misma clase de objetos genéricos o componentes. En la POO se tienen clases tales como, números, cadenas de caracteres, formas gráficas, documentos o archivos.

Clase base. Ver superclase.

Clase derivada. Ver subclase.

Comprensión del lenguaje natural. Parte del procesamiento del lenguaje natural con el objetivo de desarrollar computadoras capaces de entender las instrucciones dadas por parte del usuario de manera natural.

Conocimiento. Es la capacidad del ser humano de entender, aprender, percibir, inferir, obtener experiencia y organizar la información adquirida para dar solución a situaciones.

Conocimientos borrosos. Es información vaga, ambigua, incompleta e incluso que puede ser contradictoria, que permite almacenarse en la base de conocimientos. Se representa por las palabras casi todos, varios, muchos, pocos, frecuentemente, etc..

Crisis del software. La demanda del software ha ido crecido en todo el mundo y los desarrollos de software hechos no han cubierto de manera satisfactoria esta demanda, creando en muchos casos productos de calidad deficiente y con costos muy elevados, sin responder a la necesidad para la que se creó, de la manera deseada.

Deducción. Método de inferencia que indica un razonamiento que va de lo general a lo específico.

Demon. Procedimiento activado automáticamente si se cumple un estado específico predeterminado.

Encadenamiento hacia adelante. Estrategia de control dirigida por los datos en un sistema. Una determinada situación que es cierta actúa como disparador de una acción para llegar a una conclusión.

Encadenamiento hacia atrás. Estrategia para controlar por objetivos un sistema. Arranca de un objetivo conocido para llegar a las condiciones que dieron origen a dicho objetivo. Trabajan con una conclusión hipotética y tratan de demostrarla yendo hacia atrás para encontrar las evidencias necesarias.

Encapsulamiento. Separación de los detalles de representación de un objeto de sus detalles relevantes de la aplicación en general.

Espacio de búsqueda. Una red hecha con los nodos que representan estados del problema o del subproblema y sus ligas que representan los medios de movimientos de un nodo a otro.

Estado. Es una estructura de datos en la representación del espacio de búsqueda que da una descripción del problema desde un punto de vista particular.

Estrategias de control. Son los métodos utilizados para seleccionar el camino de inferencia a seguir en la búsqueda de la solución de problemas.

Estructura simbólica de datos. Es una estructura que se utiliza para representar conocimientos simbólicos y que también se puede almacenar en la memoria de la computadora.

Explosión combinatoria. Circunstancia presentada cuando se sobrepasa ampliamente la capacidad de la computadora para realizar un proceso en tiempo aceptable, como resultado del incremento exponencial del número de posibles soluciones de un problema, este problema puede presentarse en la búsqueda ciega.

Forward chaining. Ver encadenamiento hacia adelante.

Frames. Ver marcos de referencia.

Función heurística. Función que realiza las medidas de deseabilidad a partir de descripciones del estado del problema, las representa como números, éstos se obtienen considerando la información más relevante y la forma de evaluarla.

Grafo. Es una red de nodos conectados por arcos.

Guiones. Es una estructura tipo marco de referencia que describe secuencias de eventos que ocurren, de manera simplificada. Inventados por Roger Schank, de la Universidad de Yale, son de gran utilidad para el desarrollo de lenguaje natural en las máquinas.

Hardware. Componentes físicos que integran una computadora.

Hechos. Es una estructura de conocimientos, una aserción aceptada como verdadera, tiene un valor y/o atributo específico asociado.

Herencia. Es la capacidad de obtener información y significado de atributos, capacidades y limitaciones de nodos nuevos, marcos de referencia u objetos de uno ya existente, se pueden ligar a otros anteriores y heredar su información. En POO es la relación entre clases, en la cual la clase base o padre provee de parte de sus miembros a la clase derivada.

Heurística. El conjunto de reglas utilizadas por la **Inteligencia Artificial** para elegir entre un número grande de alternativas posibles para llegar a la solución da un problama, se basa en la experiencia y técnicas empíricas, informales y de juicios qua se hacen al conocimiento para oriantar la búsqueda.

Hill-climbing. Método de búsqueda heurística, en el que se eligen los nodos más prometedores, llevando un recorrido del árbol similar al de primero en profundidad, pero sin tener que explorar completamente el árbol, dabido a que de cada rama se van eligiendo los nodos más prometedores.

Inducción. Método de inferencia que indica un razonamianto que va de lo específico a lo general.

Inferencia. Proceso para obtener una solución a partir de evidencias previas.

Informática. Es la ciencia que se ocupa del tratamianto de la información de una manera automatizada.

Inteligencia Artificial. Campo de la computación encargado de simular el comportamiento humano inteligente.

Intérprete de reglas. Cuando se ejecuta un programa, el intérprete da reglas lo estructura dinámicamente y aplica las reglas usando un procedimiento de adaptación de modelos que verifica si las condiciones especificadas en la parte SI de una regla(visualiza las condiciones como modelos) se adaptan o coinciden con modelos similares existentes en la base de conocimiento del SE ubicada en alguna localidad de memoria de la computadora.

Jerarquía. Clasificación establecida entre las clases existentes en un sistema creado en **Programación Orientada a Objetos**, a través de la cual puede establecerse la herencia.

Las estrategias de control minimizan el espacio de búsqueda de posibles soluciones para generar sistemas eficientes de inferencia.

Lenguaje natural. Término aplicado a todo lenguaje de computación que permite que el usuario exprese su petición o problema en términos del lenguaje humano, se le ha dado este nombre para diferenciarlo del lenguaje de programación.

Lips. Medida de velocidad del funcionamiento de las máquinas de quinta generación, indica el número de inferencias lógicas realizadas por segundo.

Lisp (procesador de listas). Lenguaje de programación de **Inteligencia Artificial**, creado por McCarthy, uno de lo más populares para el desarrollo de aplicaciones inteligentes.

Lógica de predicados. Ver Cálculo del predicado.

Lógica de primer orden. Es una extensión de la lógica proposicional, que hace uso de funciones.

Lógica difusa. Es una extensión de la lógica booleana, que trata con información imprecisa.

Lógica formal. La lógica formal es otro modo común de representar los conocimientos usados generalmente en el lenguaje diario y en casi toda la matemática moderna permitiendo verificar formalmente la veracidad de la información.

Marcos de referencia. Los marcos de referencia, cuya hipótesis se debe a Marvin Minsky, son un tipo de plantilla (un modelo genérico o molde) que encuadra conjuntos de conocimientos relacionados y relativos a un tema muy concreto, que a menudo da nombre al marco de referencia. Son estructuras de datos que contienen áreas de memoria de tamaño variable y que se denominan ranuras.

Mensaje. Un mensaje es el llamado de una operación o procedimiento(lo que se denomina un método en **Programación Orientada a Objetos**) a ejecutar, establece la comunicación entre los objetos.

Metarreglas. Las reglas conocidas como metarreglas, son reglas que tratan de otras reglas y del mejor modo de aplicarlas, pero no tratan de los modos de aplicar los conocimientos, establecen las prioridades del sistema ayudando a reducir el espacio de posibles soluciones.

Método (procedimiento). Son las operaciones típicas a efectuar sobre el objeto, un objeto se define junto con un grupo de procedimientos (métodos) asociados que puede efectuar.

Nodo. Punto en una gráfica que es conectado con otros puntos a través de arcos o ligas.

Objeto. Para la **Programación Orientada a Objetos** un objeto es un paquete de código que contiene datos y procedimientos para manipularlos, que representan aspectos tangibles o intangibles de la realidad.

Paradigma. Conjunto de reglas orientadas a establecer límites y a describir cómo solucionar problemas dentro de esos límites. Cada paradigma da una forma particular de resolver un problema.

Pixeles. Elemento de visualización, es la parte más pequeña en la pantalla, que contiene miles de puntos pequeñísimos y el pixel es uno o más de ellos.

Polimorfismo. Es la habilidad de un objeto de tener métodos con el mismo nombre, pero con implementación diferente.

Pragmática. Parte del procesamiento del lenguaje natural que trata sobre las ambigüedades del mismo, para encontrar una aserción razonable, se refiere al sentido que se le da a las expresiones en la práctica, se refiere al contexto de una expresión.

Predicado. En lógica formal es una declaración sobre objetos. Sirve para expresar relaciones entre objetos y mostrar un hecho, evaluando una declaración como cierta o falsa.

Problemas simbólicos. Son los problemas de la vida diaria y las tareas cotidianas.

Procesamiento simbólico. Utilización de símbolos clasificados, más que de números, que manipulados por métodos de búsqueda para llegar a la solución de un problema.

Programación estructurada. Estilo de programación, donda los programas son hechos con un flujo definido, diseño claro y un alto grado de modularidad, existe claramente un punto de entrada y de salida definidos para cada módulo. Utiliza tres tipos de instrucciones para dirigir el flujo del programa: las secuenciales, las iterativas y las condicionales.

Programación lógica. Método de programación en el que los enunciados son de tipo lógico, es decir, representados por proposiciones o predicados lógicos.

Programación orientada a objetos (POO). Paradigma de programación que se centra en un conjunto de objetos, donde cada uno sabe como responder a un conjunto de instrucciones que se le indique, combina la abstracción de los datos, la herencia, los enlaces de comunicación y los procedimientos para manejarlos..

Prolog. Lenguaje de programación para el desarrollo de aplicaciones de IA, basado en los conceptos marcados por el cálculo del predicado.

Proposición. Una declaración lógica con un valor verdadero o falso, es una expresión atómica de la lógica.

Prueba de Turing. Juego diseñado por Turing en los años 30's, con la finalidad de dar inteligencia a una máquina de manera que actuará como una persona, teniendo otra persona que distinguir de quién se trata, si de una persona o de una máquina.

Ranura. Son áreas de memoria de tamaño variable que forman parte de un marco de referencia. Pueden contener atributos estándar como los de las bases de datos, pero además, hipótesis relativas al funcionamiento del sistema, reglas sobre determinadas situaciones de la aplicación, acciones a tomar bajo ciertas condiciones, subprogramas o apuntadores que señalan y enlazan las ranuras de un marco de referencia hacia otros marcos creando una jerarquía de relaciones que no existe en las bases de datos.

Razonamiento hacia adelante. Ver encadenamiento hacia adelante.

Razonamiento hacia atrás. Ver encadenamiento hacia atrás.

Redes neuronales. Es la simulación en computadora basada en un modelo del cerebro humano, la que debe de ser capaz de abstraer, generalizar y aprender.

Redes semánticas. Una red semántica es una notación gráfica nodo y arco que representa objetos, acciones o eventos e incorpora significados reales en el entorno exterior sobre los mismos. Los nodos representan los objetos, también pueden representar acciones o eventos. Los arcos entre los nodos representan relaciones entre los objetos(acciones o eventos) mostrados en los nodos. Los arcos que conectan los nodos de la red, muestran entre otras cosas, que esta estructura de conocimientos puede inferir información basada en un principio llamado herencia.

Reducción del problema. La representación de un problema como un conjunto de submentas o subproblemas. Se inicia con un problema y se va descomponiendo en partes o subproblemas.

Reglas de producción. Una regla es una declaración que especifica una acción que ocurre supuestamente si se dan una serie de condiciones y son la forma más común de representar el conocimiento.

Representación del conocimiento. Es una estructura de trabajo, donde se almacena y recupera cualquier información del mundo real.

Representaciones declarativas del conocimiento. Representación del conocimiento en la cual los hechos contenidos son independientes a los procedimientos o métodos que vayan a hacer uso de ellos.

Representaciones procedurales del conocimiento. Representación del conocimiento en la que se incluyen las estructuras de control que procesarán los hechos.

Reusabilidad de código. Es la habilidad de una parte de un programa o en su totalidad de ser utilizado por una nueva aplicación.

Robot. Máquina reprogramable que funciona como manipulador de objetos o materiales, utilizados ampliamente en la industria dentro de actividades repetitivas, de gran precisión y de riesgo.

Robótica. Área de la Inteligencia Artificial que se ocupa del desarrollo adecuado de robots inteligentes.

Script. Ver guiones.

Semántica. Parte de la descripción de un lenguaje de programación que trata del significado de construcciones, generalmente se basa en un lenguaje natural.

Símbolo abstracto. Símbolo cuyo significado y uso no han sido determinados, sino que depende de cada aplicación dada al símbolo.

Símbolos. Una entidad elegida para representar a una persona, objeto, concepto, operación, relación o atributo de algo del mundo real.

Sintaxis. La estructura del lenguaje de programación que es determinado por un conjunto de reglas formales.

Sistema de producción. Es el conjunto de reglas de producción referentes a un dominio que en combinación con un intérprete de reglas apoyan en la solución de problemas.

Sistemas expertos (se). Sistema computacional que aplica el conocimiento y los métodos de razonamiento para la solución de un problema en una materia especifica, de la misma forma de como lo haría un humano experto en dicha materia.

Sistemas inteligentes. Son los sistemas que combinan la tecnología de desarrollo de sistemas con los principios de la **inteligencia Artificial** para dar solución a un problema de un dominio especifico.

Slot. Ver ranura.

Software. Parte lógica de una computadora, la integra el conjunto de programas que controlan la operación de la máquina.

Solución de un problema. Es lograr un estado objetivo o meta a través de la deducción o inducción, el razonamiento o análisis procedural. La Inteligencia Artificial utiliza las búsquedas para encontrar la solución, su propósito es guiar el proceso de búsqueda en la dirección más provechosa, sugiriendo el camino.

Subclase. Se refiere a la clase creada a partir de otra previa, de la cual tomará por herencia los datos y funciones miembro que la clase base o superclase le permita. Es la clase que hereda de otra.

Superclase. Es la clase que va a compartir parte de sus datos y métodos con las clases que se deriven de la misma a través de la herencia.

Taxonomía. Clasificación hecha a un grupo de elementos.

Top-down. Diseño utilizado por una estructura jerárquica en la que sus funciones relacionadas son ejecutadas desde arriba hacia abajo, moviéndose de nivel por nivel.

Visión por computadora. Parte de la Inteligencia Artificial que se encarga de hacer dispositivos capaces de realizar la recepción, procesamiento y entendimiento de imágenes.

VLSI. (Very Large Scale Interchange), es la inclusión de cientos de miles de componentes electrónicos en un circuito integrado (chip).

Bibliografía.

1. BARR-FEIGENBAUM, Artificial intelligence Vol. I, Addison Wesley, 1989.
2. BIELAWSKI, Larry, Intelligent system design, John Wiley and Sons, U.S.A., 1991.
3. BISHOP, Peter, Fifth generation computers, Ellis Harwood, 1986.
4. COHEN, Daniel, Introduction to computer theory, John Wiley and Sons, Canadé, 1990.
5. DENNIS, Mercadal, Dictionary of artificial intelligence, VNR Computer Library, U.S.A., 1990.
6. ECKEL, Bruce, Aplique C++, McGraw Hill, México, 1991.
7. EDWARDS, Samuel, Object oriented software, Addison Wesley, 1990.
8. FIREBAUGH, , Artificial intelligence a knowledge based approach, PWS-Kent publishing Co., U.S.A., 1989.
9. GEVARTER, William, Máquinas inteligentes, Ediciones Diaz de Santos, España, 1985.
10. GOTTINGER, Hans, Artificial intelligence a tool for industry and management, Ellis Horwood, England, 1990.
11. HATON, Jean Paul, La inteligencia artificial, Paidos, 1991.
12. KATRIB, Miguel, Programación orientada a objetos en C++, Infosys, México, 1994.
13. LAFORE, Robert, Object-oriented programming in turbo C++, Waite Group Press, U.S.A., 1991.
14. LAURIERE, Jean Louis, Problem solving and artificial intelligence, Prentice Hall, 1990.
15. LINDSAY,Susan, Practical applications of expert systems, QED Information Sciences, 1988.
16. LUGER, George, Artificial intelligence and the design of expert systems, Benjemin publications Co., U.S.A., 1989.
17. MEYER, Bertrand, Object-oriented software construction, Prentice Hall, U.S.A., 1988.
18. NEGRETE, José, De la filosofía a la inteligencia artificial, Megabyte, 1992.
19. RAUCH-HINDIN Wendy B. Artificial intelligence in business, science, and industry, Prentice Hall, Inc. 1989.

20.RICH, E. Inteligencia artificial, McGraw Hill, México, 1983.

21.SCHILDT, Herbert, Utilización de C en inteligencia artificial, McGraw Hill, España, 1989.

22.SCHUTZER, Daniel Artificial intelligence an applications-oriented approach, Von Nostrand, 1987.

23.SHAPIRO, Stwart, Enciclopedia of artificial intelligence, Tomo I y II, Wiley, Canadá, 1990.

24.TAYLOR, William, What every engineer should know about artificial intelligence, MIT Press, 1989.

25.TELLO, Ernest, Object oriented programming for artificial intelligence, Addison Wesley, 1989.

26.THRO, Ellen. The artificial intelligence dictionary, Microtrend Books, 1991.

27.TURBAN, Efraim Expert systems and applied artificial intelligence, Macmillan, 1992.

28.Microsoft, Computer dictionary, Microsoft Press, U.S.A., 1991.

29.WAH, Benjamin, Computers for artificial intelligence processing, Wiley Interscience Publication, U.S.A., 1990.

30. WINBLAD, Ann, Object oriented software, Addison Wesley publishing, U.S.A., 1990.

Hemerografía.

- BRENA, Bruno Pinero, Art. La Inteligencia artificial: enfoques, herramientas y aplicaciones, Revista Soluciones Avanzadas, Sept. 1994.
- LEMAITRE, Christian Art. La inteligencia artificial en México, Revista Soluciones Avanzadas, Sept. 1994.
- WEGNER, P., Art. A multiparadigm language for object-oriented declarative programming, Revista Computer language, Julio 1995.
- LÓPEZ, Amparo, Programación orientada a objetos en C++, Revista Temas de Cómputo, UNAM-Fac. de Ciencias, 1994.

Referencias.

- [Bielawski 91]. BIELAWSKI, Larry, Intelligent system design, John Wiley and Sons, U.S.A., 1991.
- [Cohen 90]. COHEN, Daniel, Introduction to computer theory, John Wiley and Sons, Canadá, 1990.
- [D.Univ.72]. Varios, Diccionario enciclopédico universal, Tomo 4, Ediciones Credsa, España, 1972.
- [Feigenbaum 89] FEIGENBAUM, Artificial intelligence Vol. I, Addison Weslay, 1989.
- [Larousse 80]. GARCÍA, Ramón, Pequeño Larousse en color, Ediciones Larousse, España, 1980.
- [Luger 89]. LUGER, George, Artificial intelligence and the design of expert systems, Benjemin publications Co., U.S.A., 1989.
- [Micros 91]. Microsoft, Computer dictionary, Microsoft Press, U.S.A., 1991.
- [Rauch 89] RAUCH, Hindin Wendy, Artificial iIntelligence in business, science, and industry, Prentice Hall, Inc. 1989.
- [Salvat 78]. Varios, Enciclopedia Salvat, diccionario, Tomo 7, Salvat editores, México, 1978.
- [Turban 92] TURBAN, Efraim, Expert systems and applied artificial intelligence, Macmillan, 1992.
- [Wah 90]. WAH, Benjamin, Computers for artificial intelligence processing, Wiley interscience publication, U.S.A., 1990.