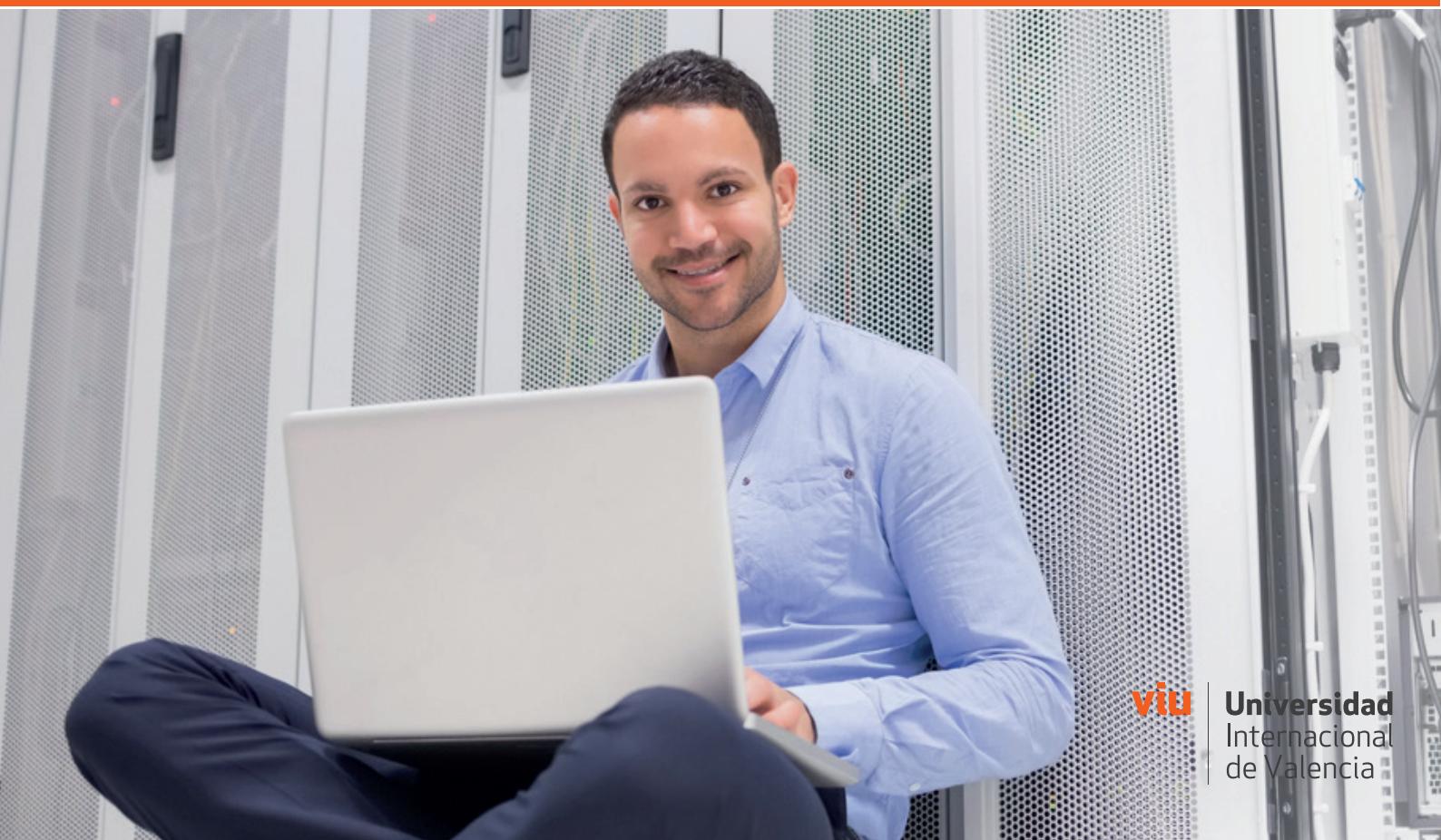


GRADO EN INGENIERÍA INFORMÁTICA

Módulo Común de la Rama de Informática

INTELIGENCIA ARTIFICIAL

Alexandra La Cruz



viu

Universidad
Internacional
de Valencia



Este material es de uso exclusivo para los alumnos de la VIU. No está permitida la reproducción total o parcial de su contenido ni su tratamiento por cualquier método por aquellas personas que no acrediten su relación con la VIU, sin autorización expresa de la misma.

Edita

Universidad Internacional de Valencia

Grado en
Ingeniería Informática

Inteligencia Artificial

Módulo Común de la Rama de Informática
6 ECTS

Alexandra La Cruz

Leyenda

abc **Glosario**
Términos cuya definición correspondiente está en el apartado “Glosario”.

 **Enlace de interés**
Dirección de página web.

Índice

1. INTRODUCCIÓN A LOS SISTEMAS INTELIGENTES	9
1.1. ¿Qué es la inteligencia artificial?	11
1.2. Fundamentos de la inteligencia artificial	12
1.2.1. Filosofía.....	12
1.2.2. Matemáticas.....	13
1.2.3. Economía	13
1.2.4. Psicología.....	14
1.2.5. Neurociencia.....	15
1.2.6. Lingüística	15
1.2.7. Ingeniería	15
1.3.Historia de la inteligencia artificial.....	15
2. AGENTES INTELIGENTES.....	17
2.1. Tipos de agentes inteligentes	18
2.2. Atributos de los agentes inteligentes.....	19
2.3. Entornos.....	20
2.4. Fuentes de incertidumbre	21
3. RESOLUCIÓN DE PROBLEMAS.....	23
3.1. Definición de un problema	25
3.2. Resolución de problemas de búsqueda de ruta	26
3.3. Soluciones de árboles de búsqueda.....	28
3.4. Estrategias de búsqueda no informada.....	29
3.4.1. Búsqueda primero en anchura.....	29
3.4.2. Búsqueda primero en anchura de costo uniforme	32
3.4.3. Búsqueda primero en profundidad	36
3.5. Estrategias de búsqueda informada	37
3.5.1. Búsqueda codiciosa primero el mejor	38
3.5.2. Búsqueda A*	39
3.6. Resolución de problemas de espacios de estados	42
3.6.1. Problema del rompecabezas de bloques deslizantes	44

4. INFERENCIA PROBABILÍSTICA	47
4.1. Inferencia por enumeración.....	49
4.1.1. Método de eliminación de variables	51
4.2. Inferencia aproximada	54
4.2.1. Métodos de muestreo directo.....	54
4.2.2. Muestreo por las cadenas de Markov (Muestreo Gibbs).....	56
5. SISTEMAS BASADOS EN REGLAS	59
5.1. Reglas	60
5.2. Motor de inferencias	60
5.3. Mecanismos de resolución.....	61
5.4. Encadenamiento de reglas.....	61
5.5. Encadenamiento de reglas basadas en objetivo	62
6. PLANIFICACIÓN.....	65
6.1. Resolución de problemas vs planificación.....	65
6.2. Planificación vs ejecución.....	66
6.3. Planificación en ambientes no deterministas	68
6.4. Planificación clásica vs planificación no-clásica.....	68
6.5. Encontrar la planificación exitosa	69
6.6. Planificación con búsqueda de espacios de estados.....	69
6.6.1. Búsqueda hacia adelante (Progresión)	69
6.6.2. Búsqueda hacia atrás (Regresión).....	69
7. PLANIFICACIÓN BAJO INCERTIDUMBRE.....	71
7.1. Planificación sin sensores.....	72
7.2. Planificación condicional.....	73
7.3. Planificación con vigilancia y replanificación.....	73
7.4. Planificación continua	73
7.5. Planificación multiagente	73
7.6. Procesos de decisión de Markov	74
8. APRENDIZAJE AUTOMÁTICO (<i>MACHINE LEARNING</i>).....	75
8.1. Preparación de los datos	78
8.2. Aprendizaje supervisado.....	79

8.3.	Aprendizaje no supervisado	82
8.4.	<i>Clustering</i>	82
8.5.	Datos de entrenamiento vs datos de prueba.....	83
8.5.1.	Validación cruzada <i>k-fold</i> (<i>K-fold cross-validation</i>).....	84
8.5.2.	Comparación de métodos de aprendizaje	84
8.5.3.	Matriz de confusión	84
8.5.4.	Curvas ROC	85
8.6.	Seleccionando el algoritmo correcto de aprendizaje	86
8.6.1.	Herramientas que te ayudan a escoger un método de aprendizaje.....	87
8.7.	Aprendizaje por refuerzo	88
8.8.	Aprendizaje profundo	89
9.	MODELO DE MARKOV.....	93
9.1.	Modelo oculto de Markov (HMM)	94
9.2.	Arquitectura de un modelo oculto de Markov	95
9.3.	Estimación de los parámetros en un modelo oculto de Markov.....	96
9.4.	Algoritmo de Viterbi.....	97
10.	APLICACIONES.....	99
10.1.	Juegos	99
10.2.	Visión por computadora o visión artificial	100
10.2.1.	Formación de las imágenes	101
10.2.2.	Proyección en perspectiva.....	102
10.2.3.	Punto de fuga	103
10.2.4.	Imagen digital.....	103
10.2.5.	Extracción de características	107
10.2.6.	Filtros lineales.....	109
10.2.7.	Filtro gausiano	110
10.2.8.	Detector de bordes de Canny	110
10.2.9.	Detector de esquinas de Harris	111
10.2.10.	Detector de características actuales	112
10.3.	Robótica.....	113
10.3.1.	Clasificación de los robots.....	115
10.3.2.	Clasificación de los robots según su arquitectura	115

GLOSARIO.....	117
ENLACES DE INTERÉS.....	119
BIBLIOGRAFÍA.....	121

1. Introducción a los sistemas inteligentes

La finalidad del curso es ofrecer al estudiante una visión del mundo de la Inteligencia Artificial (**IA**). Definiendo a los agentes inteligentes como sistemas que pueden tomar decisiones y acciones ante una situación dada. Se plantean temas relacionados a la resolución de problemas, estudiando los métodos de decisión ante situaciones diferentes, planificando los pasos con antelación para resolver problemas similares a los del ajedrez. Además de definir las formas de representación del conocimiento y cómo razonar de forma lógica dado dicho conocimiento. Así mismo se describen los distintos métodos de aprendizaje para la toma de decisiones: automático, supervisado y no supervisado. Adicionalmente se describe la forma como se comunican, perciben y actúan los agentes inteligentes; bien sea mediante la visión, el tacto, la audición o la comprensión del idioma. Finalmente, se concluye analizando el pasado, la evolución y el futuro de la inteligencia artificial tomando en cuenta aspectos éticos, filosóficos, culturales y económicos.

En este manual trataremos también los puntos básicos que aparecen en el libro de Stuart Russel y Peter Norvig (2010), *Artificial Intelligence, A modern Approach* en su tercera y más reciente edición. Libro ampliamente utilizado en IA como libro de referencia, es prácticamente la biblia de la inteligencia artificial.

Hoy en día se habla de que estamos viviendo la revolución industrial 4.0 y ¿A qué se refiere esto? Para responder a esta pregunta, describamos como ha sido la evolución de la revolución industrial.

La **primera revolución industrial** se inició en la segunda mitad del siglo XVIII (~1740-1840), y la creación de máquinas de vapor y la mecanización de muchos procesos permitió transformar la economía rural en una economía industrializada.

La **segunda revolución industrial** (~1850-1914) se inicia con el uso de nuevas fuentes de energía como la electricidad, el gas y el petróleo.

La **tercera revolución industrial**, también llamada **revolución tecnológica**, iniciada a partir de 1970 aproximadamente, consistió en el uso de las tecnologías de la información y la comunicación.

Hoy en día, a partir de 2010 aproximadamente se ha iniciado lo que se denomina “la **revolución industrial 4.0**” (ver Figura 1), en la que los sistemas computacionales están siendo interconectados entre sí, entre otras cosas. Esto ha permitido la generación y compartición de grandes cantidades de datos que están siendo manejados de manera inteligente con el fin de entender, comprender, analizar, reconocer, razonar y predecir el comportamiento de los sistemas y de los datos.

Un interesante libro para leer y conocer en profundidad lo que es la revolución industrial es el libro *La Cuarta Revolución Industrial* de Klaus Schwab (2016).

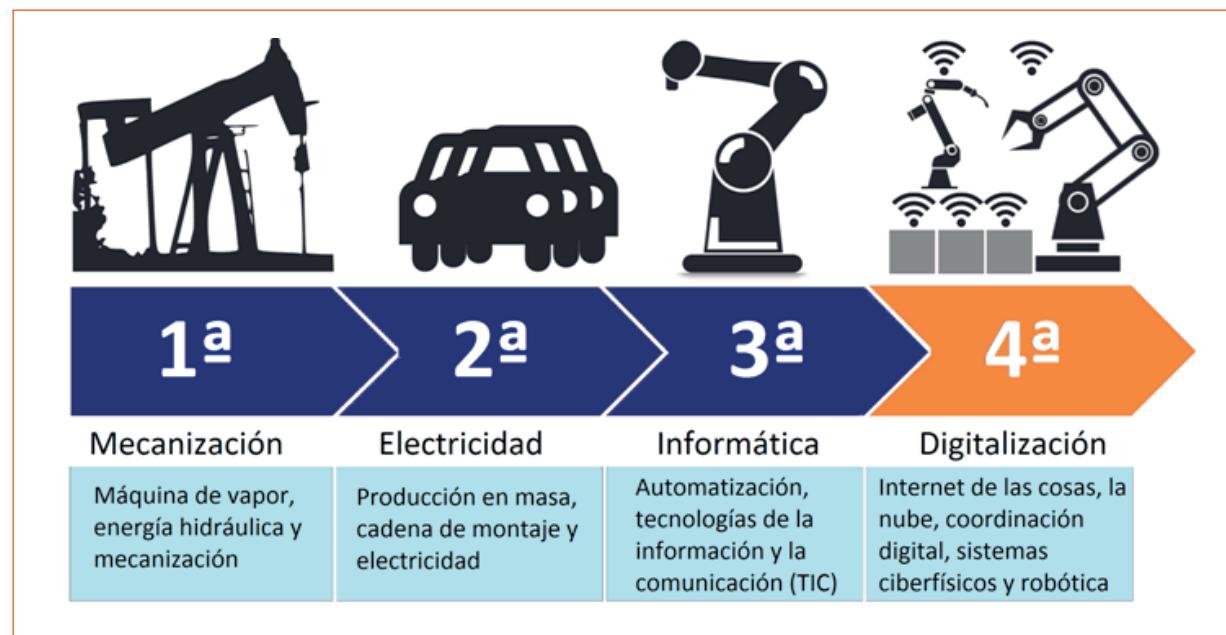


Figura 1. Revolución Industrial. Describe cómo se ha llegado a la llamada revolución industrial y qué significa. Recuperado de <https://revolucionindustrialmbaee.wordpress.com/revolucion-4-0/>.

La IA es el motor de la revolución industrial 4.0. Es un área interdisciplinaria que va avanzando de manera acelerada con la creación, uso e interconexión de las nuevas tecnologías. La revolución industrial 4.0, basada fundamentalmente en el uso de técnicas de IA, busca la transformación digital de las empresas a fin de convertirlas en organizaciones inteligentes y conseguir los mejores resultados de negocio. Muchos expertos dicen que dentro de poco tiempo los negocios que no sean transformados digitalmente, simplemente se quedarán en el pasado y seguramente desaparecerán. Sin embargo, esto es un tema que se sigue discutiendo. Aunque muchos autores hablan de las consecuencias de esta revolución industrial aún hay mucha especulación al respecto. Se habla de la desaparición de

muchos empleos, al mismo tiempo que se habla de la creación de nuevas profesiones. Así que aún hay mucho por discutir, descubrir, estudiar, analizar y observar.

1.1. ¿Qué es la inteligencia artificial?

Muchos autores definen la inteligencia artificial en términos generales como el conjunto de métodos, procesos o algoritmos computacionales que permiten la resolución de problemas de modo inteligente. ¿Y qué significa esto? ¿Qué significa eso de "modo inteligente"? Según la psicología se define la inteligencia como la facultad o habilidad de la mente humana que le permite aprender, entender, analizar, razonar y tomar decisiones. Computacionalmente, hoy en día se utiliza la inteligencia artificial para desarrollar sistemas que sean capaces de aprender, entender, analizar, razonar, tomar decisiones correctas a partir de los datos de entrada o del estado actual o situación dada. Incluso permite a los sistemas predecir comportamientos futuros de los datos o del sistema.

Alan Turing es conocido como el padre de la inteligencia artificial y de la computación. Turing definió lo que se llama el **test de Turing** o lo que en un principio Turing llamó el **juego de la imitación**. El test de Turing fue un método creado por Alan Turing para responder científicamente a si una máquina puede o no pensar por sí misma.

El test de Turing consistió en desarrollar una conversación entre un humano y una máquina diseñada para interactuar a través de un chat. La idea era determinar si se conversaba con un humano o una máquina. La conversación se realizaba en cuartos separados, el humano en prueba no sabía con quién estaría conversando. La conversación tenía una duración de 5 minutos y la idea consistía en que la máquina debía convencer a la persona con quien chateaba de que se trataba de una persona. Si la convicción entonces ganaba la máquina (Russell & Norvig, 2010).

La IA también explica por medio de las matemáticas los patrones que existen en la naturaleza, como la manera como obtiene el pez zebra sus rayas o las manchas de los leopardos, entre otras cosas aún más asombrosas.

La IA ha sido definida a lo largo del tiempo por diferentes autores como:

- *Estudio de la computación que observa que una máquina sea capaz de percibir, razonar y actuar.* (Winston, 1992).
- *Ciencia de la obtención de máquinas que logren hacer cosas que requerirían inteligencia si las hiciesen los humanos.* (Minsky y Papert, 1969).
- *Nuevo esfuerzo excitante que logre que la computadora piense, máquinas con mentes, en el sentido completo y literal.* (Haugeland, 1985).
- *Rama de la ciencia computacional preocupada por la automatización de la conducta inteligente.* (Luger, 1995).

Diferentes áreas del conocimiento han contribuido a lo que hoy en día conocemos como inteligencia artificial. Entre ellas la filosofía, las matemáticas, las ciencias de la computación, la psicología, la

lingüística, la economía y la neurociencia. A continuación, se describe brevemente cuál ha sido la contribución de cada una de estas áreas al tema de la IA.

1.2. Fundamentos de la inteligencia artificial

Describiremos históricamente cómo diferentes disciplinas han contribuido con sus ideas, puntos de vista y técnicas al área de la IA, a partir de preguntas que surgieron en su época. Sin embargo, tal como Russel y Norvig (2010) acotan en su libro, no se pretende con ello dar la impresión de que sean las únicas preguntas e ideas que dichas disciplinas usaron o introdujeron como fundamentos de la IA.

1.2.1. Filosofía

Filósofos como Sócrates, Platón, Aristóteles y Leibniz fueron quienes sentaron las bases para la inteligencia artificial.

Aristóteles (384-322 A.C.) fue uno de los primeros en describir un conjunto de reglas que explican la forma como el ser humano produce conclusiones a partir de un grupo de premisas, razonar a partir de un conjunto de reglas. Luego Ramon Llull (1232-1315), Thomas Hobbes (1588-1679), Leonardo da Vinci (1452-1519), Wilhelm Schickard (1592-1635), Blaise Pascal (1623-1642) y Gottfried Wilhelm Leibniz (1646-1716) entre otros, trabajaron en tratar de definir y descifrar la manera de funcionar del razonamiento humano a partir de un conjunto de premisas o reglas.

Luego, René Descartes (1596-1650) abre el tema sobre ver la mente humana como un sistema físico y propone hacer una distinción entre la mente y la materia. Entramos entonces en las corrientes materialistas y dualistas. Descartes era un defensor del dualismo. El dualismo define a la conciencia como algo superior e intangible, no físico, que puede ser visto como lo que denominamos el alma. Es decir que la conciencia proviene de algo que es mayor a la suma de sus partes. Mientras que el materialismo, lo ve como que el todo emerge de la suma de sus partes, no deja espacio para el libre albedrio. Algunos autores consideran al materialismo contraproducente, pues la materia no piensa, no razona, no actúa a voluntad; entonces ¿cómo puede razonar nuestra mente? Considerando que todo lo que hacemos es parte de una naturaleza interna.

Hasta ahora se había definido el razonamiento a partir de premisas y se había considerado la mente como un sistema físico que gestiona conocimiento, pero ¿De dónde viene dicho conocimiento? Contribuciones de Francis Bacon (1561-1626), John Locke (1632-1704) y el movimiento empírico con David Hume (1711-1776) quien define el principio de inducción, Rudolf Carnap (1891-1970) y la doctrina del positivismo lógico, que establece que todo el conocimiento se puede caracterizar a través de teorías lógicas relacionadas a partir de la observación correspondientes a estímulos sensoriales. Por otro lado, Carnap y Carl Hempel (1905-1997) son quienes definen un procedimiento computacional explícito para la extracción de conocimiento a partir de experiencias primarias. Fue quizás la primera teoría en mostrar la mente como un proceso computacional.

Finalmente nos queda definir la relación entre el conocimiento y la acción, vital para la IA dado que la inteligencia requiere de acción y razonamiento. El hecho de comprender cómo se generan las acciones a partir del conocimiento permite definir agentes cuyas acciones sean “racionales”.

Aristóteles argumentó por ejemplo que las acciones se pueden justificar con la conexión lógica entre los objetivos y el conocimiento. Son entonces Aristóteles, Antoine Arnauld (1612-1694) y John Stuart Mill (1806-1873), entre otros, quienes contribuyeron a justificar las acciones a partir del conocimiento y a la teoría de decisión racional.

1.2.2. Matemáticas

Las matemáticas la han provisto de herramientas para el manejo de la lógica, certidumbre, incertidumbre y modelos probabilísticos, así como el cálculo numérico han permitido modelar fenómenos y manejar el razonamiento algorítmico. Pasamos entonces de las ideas determinadas filosóficamente a la ciencia formal y formulación matemática de las mismas. La lógica formal se remonta desde la antigua Grecia. Sin embargo, no es hasta mucho más tarde que George Boole (1815-1864) define la lógica Booleana y luego Gottlob Frege (1848-1925) crea la lógica de primer orden. Alfred Tarski (1902-1983) muestra cómo relacionar objetos de una lógica con objetos del mundo real. Para entonces se definieron reglas formales y herramientas para llegar a conclusiones válidas y poder representarlas computacionalmente.

Se desarrolla entonces lo que se conoce como el primer algoritmo computable no trivial, el algoritmo Euclídeo, que calcula el máximo común divisor a partir de restas sucesivas, algoritmo descrito en Shoup (2008). En 1931, Kurt Gödel (1906-1978) define el *Teorema de incompletitud* donde demostró ciertas limitaciones sobre algunas teorías aritméticas que no son posibles de demostrar y al mismo tiempo no se pueden refutar o resultan contradictorias y por lo tanto son indecidibles a partir de razonamiento matemático. Demostrando que en cualquier lenguaje con capacidad para expresar axiomas matemáticos existen aseveraciones imposibles de validar algorítmicamente según Russell y Norvig (2010). Esto lleva a Alan Turing (1912-1954) a intentar caracterizar aquellas funciones characterizables. Crea la máquina de Turing, de la cual Turing demuestra que ninguna máquina puede decidir si producirá o no resultados a partir de unas entradas.

De allí surge el término **intratable**, la máquina dice que un problema es intratable en la medida en que el tiempo para su resolución de casos particulares aumenta exponencialmente al tiempo que aumenta el número de casos. Esto conlleva a tratar problemas intratables dividiéndolos en varios subproblemas tratables. Surge entonces la teoría de los **NP-completos**, que engloba aquellos problemas de complejidad **NP** donde no es posible encontrar una solución sino realizando una búsqueda exhaustiva de soluciones posibles o casos posibles. La IA ha permitido explicar en algunos casos que existen problemas NP-completos difíciles de resolver y otros fáciles.

La probabilidad es otra de las contribuciones importantes de las matemáticas y contribuye en gran medida en la IA, permitiendo hacer mediciones en presencia de incertidumbres y teorías incompletas. Las teorías Bayesianas son importantes para el razonamiento incierto de sistemas de IA.

1.2.3. Economía

La economía en temas como la teoría de la decisión, la teoría de juegos y los procesos de decisión de Markov, son conocimientos que contribuyeron a la IA con los modelos para la toma de decisión. Al fin y al cabo, la economía se refiere a la toma de decisiones que permite obtener finalmente

beneficios. Se refiere en parte a cómo llevar a cabo el proceso de toma de decisiones de tal manera que maximice el rendimiento de los sistemas. La teoría de decisión combina la probabilidad con la utilidad proporcionando el marco para la toma de decisiones bajo incertidumbre. León Walras (1834-1910) fue quien formalizó los principios matemáticos para medir la utilidad y luego fueron mejorados por Frank Ramsey (1931), John von Neumann y Oskar Morgenstern (1944). En economías grandes se utiliza ampliamente la teoría de decisión, sin embargo, en economías complejas se utiliza la teoría de juegos. Campos como la investigación de operaciones y su utilidad en problemas de optimización han contribuido igualmente a la IA.

Herbert Simon (1916-2001) premio nobel de economía y pionero en el campo de la IA, estudió los modelos basados en satisfacción, modelos que permiten tomar decisiones suficientemente buenas en vez de decisiones óptimas, proporcionando un modelo más cercano al comportamiento humano.

1.2.4. Psicología

La psicología y sus teorías en el conductismo o psicología cognitiva han permitido definir los modelos mentales y procesos cognitivos que han servido para modelar el comportamiento humano. Su contribución ha sido en áreas que intentan analizar y razonar cómo piensan y actúan los humanos. El conductismo estudia el comportamiento humano en respuesta a estímulos, mientras que la psicología cognitiva estudia lo referente a los procesos mentales implicados en el conocimiento por medio de tres importantes pasos:

- Atención a través del estímulo.
- Cómo se percibe a través de los sentidos.
- Cómo se almacena en memoria, permitiendo definir procesos mentales en el individuo, que le permiten realizar una acción según el conocimiento obtenido en épocas anteriores.

Basándose en la psicología cognitiva, Craik (1967) establece tres elementos claves para diseñar un agente basado en conocimiento:

- El estímulo deberá ser traducido a una representación interna.
- Dicha representación se debe manipular mediante procesos cognitivos a fin de generar nuevas representaciones internas.
- Estas, a su vez, se traducirán en acciones.

Es en 1956 cuando en el MIT se inicia la creación del campo de ciencia cognitiva. Los trabajos presentados por Miller (1956) "The Magic Number Seven", Chomsky (1956) "Three Models of Language", y Allen Newell y Herbert Simon (1957) "The Logic Theory Machine" demostraron que se podían utilizar los modelos informáticos para modelar la psicología de la memoria, el lenguaje y el pensamiento lógico, respectivamente.

1.2.5. Neurociencia

La Neurociencia, por otra parte, ha contribuido a la IA con los conocimientos relacionados con la forma en que el cerebro procesa la información y los procesos mentales implicados en el comportamiento humano. Saber cómo se generan los pensamientos en el cerebro sigue siendo uno de los grandes misterios. Este es uno de los grandes retos de la IA hoy en día.

1.2.6. Lingüística

Es importante la lingüística y sus conocimientos para el desarrollo de técnicas de procesamiento del lenguaje natural o lingüística computacional. Para la IA se toma como modelo la interacción verbal humana, hacer reconocimiento y generar la respuesta hablada.

1.2.7. Ingeniería

Ciencia de la computación ha contribuido grandemente a la IA gracias a los avances tecnológicos en cuanto al procesamiento, los tiempos para el desarrollo de métodos de IA se han acortado y ha permitido el crecimiento que esta área tiene hoy en día. La ingeniería ha permitido modelar y desarrollar el comportamiento inteligente de los agentes, así como el diseño y creación de los dispositivos inteligentes, o dispositivos que permiten extraer parámetros de los dispositivos no inteligentes.

1.3. Historia de la inteligencia artificial

Intentaremos describir su historia mediante un breve resumen histórico de varios artículos expuestos en la web (Berzal, 2017), y del libro de Russell & Norvig (2010). Aunque se dice que la IA tiene sus inicios en los años 30 con Alan Turing, no es sino en 1950 cuando Turing publica su artículo sobre el famoso Test de Turing en la revista Mind con su artículo “Computing Machinery and Intelligence” y de donde parte de su hipótesis: **¿Pueden las máquinas pensar?** A él se le atribuye ser el padre de la IA, su test aún sigue estando en vigencia y ha sido motivo de muchos estudios.

En los años 40 y 50 se desarrollaron programas con tareas básicas de razonamiento como jugar al ajedrez, a las damas, resolver teoremas geométricos... El juego de Damas fue el primer juego de IA desarrollado, sin embargo, con él se podía tanto ganar como perder, no era infalible. McCulloch y Pitts (1943) desarrollan el primer circuito booleano como un modelo del cerebro.

Según algunos estudios se considera el año 1956 como el año de inicio de la IA moderna y se considera como padres de esta nueva era de la IA a John McCarty, Marvin Minsky y Claude Shannon quienes acuñaron a la IA el nombre de “la ciencia e ingenio de hacer máquinas inteligentes especialmente programa de cálculo inteligente” durante la conferencia *Darmouth Summer Research Project on Artificial Intelligence*.

Más tarde, en los años 60 y 70 y con el desarrollo de algoritmos para el razonamiento lógico Robinson y estudios sobre complejidad computacional, nuevamente desciende el uso de redes neurales hasta casi a su desaparición. Hoy en día, con el desarrollo de aplicaciones de Deep Learning la IA ha tenido

un impulso importante y es que la cantidad de datos e información que se manejan, junto con los desarrollos de tecnología de *hardware* cada vez más poderosos y asequibles lo hacen posible. Durante este período se desarrollan también en paralelo los sistemas basados en conocimiento y manejo de lenguaje natural. Entre algunos de los ejemplos de los primeros sistemas basados en conocimiento tenemos a Mycin, inspirados en Dendral. Es un sistema utilizado para el diagnóstico de enfermedades infecciosas de la sangre. Dendral era utilizado en la industria química para interpretar la estructura molecular de algunos componentes (Lederberg, 1987).

Durante los años 80 y 90 nacen los sistemas expertos y esto hace que los modelos de redes neuronales bajen en popularidad, a partir de allí se dice que inicia el invierno para la IA. Después, en 1986, las redes neuronales retoman su popularidad con el desarrollo de modelos conexionistas como el modelo de *backpropagation*. Luego nacen las aplicaciones SAT como solucionadores de problemas de satisfacción de restricciones y los modelos ocultos de Markov para el procesamiento del lenguaje natural.

Más recientemente (IBM, 2011) cuenta con la existencia de la supercomputadora Watson de IBM. Watson es una computadora que es capaz de aprender a partir de la acumulación de información, utiliza el lenguaje humano como lenguaje natural para interactuar y dichas interacciones le permiten acumular más información, conectarlas y aprender. Watson ganó el concurso televisivo *Jeopardy* de Estados Unidos. El concurso consiste en realizar preguntas de todo tipo sobre cultura y conocimiento. Watson participa con dos de los mejores concursantes del programa y resulta ganador.

2. Agentes inteligentes

Un agente inteligente en IA (ver Figura 2) es un elemento de *software* desarrollado para realizar la mejor acción posible ante una situación dada. **Un agente inteligente debe tener la capacidad de estimar varios valores que le permitan maximizar la utilidad de sus acciones y a partir de allí ejecutar la acción y seguir adelante.** En términos generales un agente inteligente es un ente cuyo objetivo es resolver un problema bien definido, realizando acciones que modifiquen el estado actual según sus objetivos.

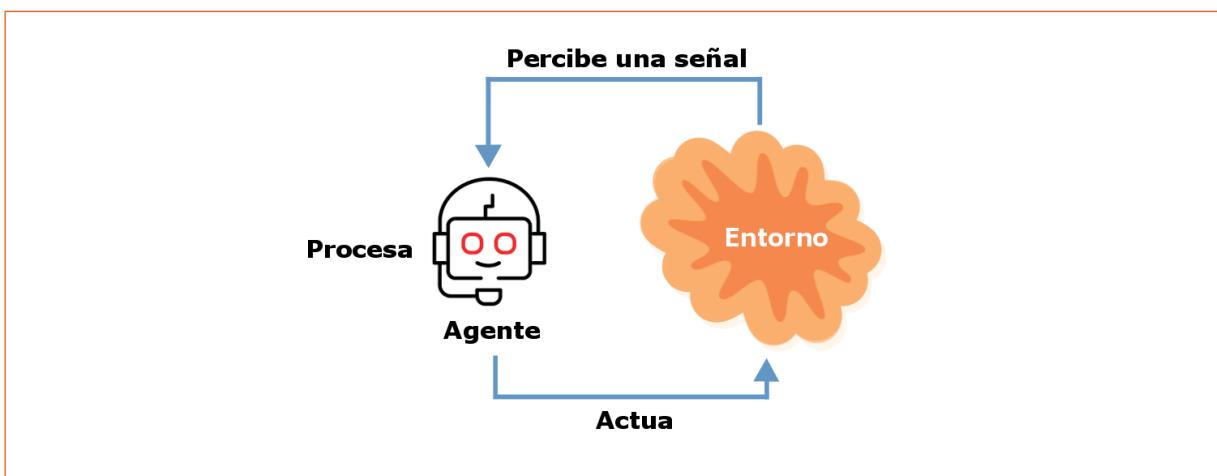


Figura 2. Ilustración de un agente inteligente.

Un agente inteligente percibe información acerca del estado actual en su entorno, valida si es el estado objetivo, y si no lo es, evalúa y busca ejecutar una acción que le lleve a cumplir su objetivo (ver Figura 2). Los agentes inteligentes deben ser dotados inicialmente de conocimientos específicos para poder tener claro cuáles son los posibles escenarios o estados y cuál es el objetivo. Generalmente suponen una percepción y una ejecución ideal. Seguir la pista del estado del entorno es una de las tareas y capacidades principales de un agente inteligente.

Según Russell y Norving (2010), "Un agente inteligente es cualquier cosa que puede ser vista como algo que percibe su ambiente a través de sensores y actúa sobre su ambiente a través de efectores".

A diferencia de un *software* tradicional, un agente inteligente debe tener cierta autonomía para realizar sus tareas o acciones que le lleven a cumplir su objetivo. Requiere además del uso de razonamiento y aprendizaje entre otras técnicas que le capacitan para interpretar la información o estado actual, según obviamente el conocimiento al cual tenga acceso. Se puede decir que hay tres niveles de inteligencia en un agente inteligente:

- **Baja:** permite al usuario expresar sus preferencias, por ejemplo, el caso de los *adWords* en Google. Se le presenta la posibilidad, al usuario, de querer recibir los anuncios o de recibirllos, en cuyo caso será información que retroalimente a los agentes publicitarios inteligentes de Google.
- **Mediana:** la toma de decisiones para realizar una acción viene dada por un proceso de inferencia basado en un conjunto de reglas de razonamiento combinadas con el conocimiento al cual se tenga acceso.
- **Alto:** este nivel es de los más interesantes y hacia donde se quiere llegar en el área de la aplicación de IA. Este nivel superior del agente es la capacidad del agente de mejorarse a sí mismo, generar nuevos conocimientos y nuevas formas de racionamiento o conexión entre el conocimiento manejado. En otras palabras, aprender, de la misma forma que puede hacerlo un humano.

2.1. Tipos de agentes inteligentes

Se puede decir que existen tres tipos principales de agentes inteligentes:

- **Agentes reactivos (estímulo - acción):** calculan las acciones directamente a partir de lo que perciben del entorno, como estado actual. Este tipo de agente generalmente sigue un enfoque denominado conexiónista, en muchos casos permite generar rápidamente acciones "correctas".
- **Agentes proactivos:** este tipo de agente tiene la capacidad de evaluar sus posibles acciones y sopesarlas, anticipándose a los efectos potenciales y probables de cada acción. Esto le permite ejecutar acciones menos equivocadas, más acertadas e irrevocables.
- Por último, tenemos los **agentes híbridos:** combinan ambos enfoques, el enfoque reactivo para acciones inmediatas y el enfoque proactivo para acciones estratégicas.

A partir de estos tipos se tienen algunos tipos más especializados de agentes:

- **Agentes reactivos simples:** responden directamente a las percepciones del estado.

función AGENTE-REACTIVO-SIMPLE(percepción) **devuelve** una acción
reglas = {conjunto de reglas}

```
estado ← interpreta-entrada(percepción)
regla ← regla-coincidencia(estado, reglas)
acción ← regla-accion(regla)
end
```

- **Agentes reactivos basado en modelos:** mantienen un estado que les permite conocer los aspectos del mundo que no son perceptibles.

función AGENTE-REACTIVO-Modelo(percepción) **devuelve** una acción
estado = {estado, regla, acción}

```
estado ← actualizar-estado(estado, acción, percepción)
regla ← regla-coincidencia(estado, reglas)
acción ← regla-accion(regla)
end
```

- **Agentes basados en objetivos:** actúan en función de alcanzar su objetivo, constantemente se preguntan si ya obtuvieron su objetivo, después de ejecutar cada acción y conocer el estado actual en el que se encuentra, para replanificar en algunos casos la próxima acción a seguir.
- **Agentes basados en utilidad:** actúan en función de maximizar la utilidad de sus acciones.
- **Agentes que aprenden:** utilizan mecanismos de aprendizaje, estructuración del conocimiento combinado con el conjunto de reglas de razonamiento.

2.2. Atributos de los agentes inteligentes

Un agente inteligente está caracterizado por una serie de propiedades que dependen de las tareas a cumplir, lo que se espera o para lo que fue diseñado inicialmente. Esto tiene que ver principalmente con la definición de un agente inteligente: ente, dispositivo o programa computacional con la capacidad de actuar de manera autónoma y flexible en el entorno en el cual actúa. La característica de flexible se refiere a la forma de actuar ante el estado actual del entorno, si es un agente reactivo o proactivo.

Entre los atributos o características de un agente inteligente tenemos:

1. **Autónomo:** debe ser completamente autónomo, capaz de decidir las acciones a realizar basándose en su experiencia. Debe ser capaz además de adaptarse a cambios suaves o severos de su entorno.
2. **Sociable:** debe ser capaz de comunicarse en el entorno con otros agentes o entidades.
3. **Racional:** el agente siempre realiza lo que él cree que es lo correcto a partir de los datos que percibe del entorno.

4. **Reactivo:** actúa según lo que percibe de su entorno, sus acciones son motivadas por la información que recibe de su entorno.
5. **Proactivo:** debe ser capaz de controlar sus propios objetivos a pesar de los cambios no esperados en el entorno.
6. **Adaptable:** debe ser capaz de adaptarse a los cambios, esto está muy relacionado con la capacidad de aprender que permite generar cambios en su comportamiento basándose en el aprendizaje obtenido, en su experiencia de acciones.
7. **Tener movilidad:** tiene que ver con la capacidad del agente de trasladarse a través de una red telemática.
8. **Veraz:** se asume que toda la información transmitida por un agente inteligente es veraz, no engaña a propósito.
9. **Benevolente:** se asume que un agente inteligente está dispuesto a ayudar a otros agentes en la medida en que sus acciones benevolentes no entren en conflicto con sus propios objetivos. Por ejemplo, las competencias robóticas de sumo, su objetivo es sacar al oponente del espacio de lucha, en este caso no puede ser benevolente, pues su objetivo es luchar contra su adversario.

2.3. Entornos

Los entornos en los que funciona un agente inteligente pueden variar según la característica del problema o definición del mismo entorno. El entorno puede ser:

1. **Ambientes observables parcial o completamente:** un ambiente completamente observable puede ser el mapa en un sistema de búsqueda de la ruta más cercana al objetivo. Un laberinto, por ejemplo. En un ambiente completamente observable se tiene una visión maestra del entorno que rodea al agente. Un laberinto, el juego de ajedrez, las damas, etc.
 - Un ambiente parcialmente observable es posible que solo tenga la información más cercana al agente más que la información más cercana al objetivo. En cuyo caso, la estrategia en muchos casos puede ser ensayo y error, o la búsqueda de máxima utilidad.
2. **Ambientes estocásticos vs determinísticos:** en un ambiente estocástico está el factor azar y la sorpresa, en algunos casos con un peso probabilístico conocido o desconocido al tomar una u otra acción.
 - En un ambiente determinístico no se cuenta con el factor sorpresa y se tiene claro qué cosas afectarán a cada acción del agente. En ambientes determinísticos los resultados de las acciones son predecibles.
3. **Ambientes continuos vs discretos:** en ambientes discretos se tiene un conjunto finito de percepciones y acciones (ajedrez), mientras en ambientes continuos (vehículos autónomos) se tienen un conjunto de percepciones y acciones infinitas, continuas.

4. **Ambientes estáticos vs dinámico:** en ambientes dinámicos el entorno cambia cuando el agente está decidiendo qué acción tomar, en cuyo caso, el agente debe estar pendiente de los constantes cambios del entorno, mientras que en el estático no sucede esto.
5. **Individual o multiagentes:** en los entornos multiagentes encontramos ambientes competitivos o colaborativos, en los que los agentes o están compitiendo entre sí o están colaborando en la ejecución de alguna tarea, en cuyo caso tienen un objetivo en común.
6. **Naturaleza competitiva:** son los casos en los que se tiene uno o más adversarios, que pueden ser otros agentes inteligentes o un humano.

2.4. Fuentes de incertidumbre

La incertidumbre es de esos aspectos, hechos o estados en los que le impide a un agente inteligente saber con certeza cuál debería ser la acción a ejecutar. Un agente inteligente en teoría debe ser capaz de conocer suficientes hechos sobre su entorno, esto le permite ejecutar acciones que garanticen su desarrollo o cumplir sus objetivos. Sin embargo, no siempre es así, no siempre tiene toda la verdad acerca del entorno. La incertidumbre se puede presentar por varios factores:

1. **Limitado número de sensores:** en el caso en que no sea posible disponer de los sensores necesarios para obtener toda la información acerca del estado del entorno. En algunos casos se pueden crear parámetros al azar o evaluando su probabilidad de ocurrencia.
2. **Adversarios:** cuando se tiene un ambiente multiagente, no siempre se pueden predecir las acciones del adversario, hay un factor de incertidumbre que puede variar según el caso, la situación o problema.
3. **Ambientes estocásticos cambiantes:** en ambientes estocásticos existe un factor de incertidumbre que dependerá de la distribución de la probabilidad de los parámetros que modifican el estado del entorno
4. **Flojera:** en este caso es el agente que no tiene todas las capacidades para validar su entorno.
5. **Ignorancia:** cuando no se tiene conocimiento de los cambios en el entorno.

3. Resolución de problemas

La resolución de problemas tiene que ver con la teoría y las técnicas para construir agentes inteligentes que puedan planificar acciones y resolver problemas. En particular se describirá la forma de resolver problemas cuya complejidad radica en que hay varios estados u opciones posibles. **Estamos hablando de agentes inteligentes basados en objetivo, específicamente de los agentes resolventes de problemas.** Estos agentes son agentes cuya función es conseguir una secuencia de acciones que permita obtener los objetivos deseados o estados deseables.

Por ejemplo: un problema de navegación donde se puede elegir entre varias rutas para llegar a un punto. El problema en este caso radica en elegir el camino correcto (el óptimo, la ruta más cerca, o el de menos costo) al iniciar el recorrido, y en cada una de las intersecciones siguientes hasta llegar al punto de destino. Lo que se traduce en definir una secuencia de acciones que me lleven de un punto **A** a un punto **B**. Una dificultad adicional o complejidad diferente puede ser también tener que resolver el problema de navegación de ir de un punto **A** a un punto **B**, en un entorno que es parcialmente observable, en el cual no podemos ver todas las rutas posibles y podríamos no ver los resultados de las acciones tomadas o incluso las acciones posibles son desconocidas.

Antes, definamos qué son los agentes resolventes de problemas. Son agentes que tienen un objetivo. Lo primero es considerar formular el objetivo. En nuestro ejemplo, el agente se encuentra en la ciudad de Arad, de vacaciones y quiere llegar a la capital de Rumanía, Bucarest. Él ha alquilado un vehículo y quiere conocer la mejor ruta a seguir para llegar a Bucarest (ver Figura 3). Lo primero que hace es

comprar un mapa de **rutas Michelin** y estudiar el mapa. El objetivo en este caso es llegar a Bucarest. Así que puede rechazar aquellas acciones que no le lleven a Bucarest y enfocarse solo en las que sí le permitan llegar a su objetivo. De lo contrario podría ir a cualquier lugar sin ningún objetivo claro.

Después de la formulación de objetivos se debe formular el problema, definiendo el conjunto de acciones y estados que se deben considerar y que pueden llevar al agente a cumplir con el objetivo. Este proceso de conseguir ese conjunto de acciones y estados se denomina **búsqueda**. Una vez hallada una solución se procede a la fase de **ejecución**. Es así como el diseño del agente comprende de tres fases: **formular, buscar y ejecutar**.

Otra consideración importante de los agentes resolventes de problemas es que se tratan de entornos:

- a. Estático (no ocurren cambios en el entorno).
- b. Observable (se debe tener el mapa de rutas, de caminos para llegar de un punto a otro).
- c. Discreto.
- d. Determinístico (no hay aleatoriedad en las acciones o estados).



Figura 3. Mapa de Rumanía. Recuperado de <https://erumania.wordpress.com/tag/mapas-rumanas/>

Definamos entonces qué es un problema a resolver por los agentes resolventes.

3.1. Definición de un problema

Definamos ahora formalmente un problema. Un problema se compone de varios elementos:

1. **El estado inicial:** sería el punto de partida del problema a solucionar por el agente. Serían las condiciones iniciales que definen el punto de partida para el agente.
2. **Las acciones:** son funciones en las que, dado un estado de entrada, retorna un conjunto de posibles acciones que el agente puede ejecutar partiendo del estado actual.

$$\text{Acción } (S) \rightarrow \{a_1, a_2, \dots, a_i\}$$

En algunos problemas el agente puede tener las mismas acciones disponibles en todos los estados. En otros casos se podrán tener diferentes acciones dependientes del estado actual. Cada una de estas acciones lleva al agente a un nuevo estado. Esto nos lleva a definir una ruta, **una ruta es una secuencia de estados conectados por una secuencia de acciones.**

$$\text{Camino} = \{ S^0 \xrightarrow{a_1} S^1 \xrightarrow{a_p} \dots \xrightarrow{a_q} S^j \}$$

Así mismo podemos definir la función resultado como una función cuya entrada es un estado y una acción, que retorna un nuevo estado como resultado de tomar la acción dada.

$$\text{Resultado}(S, a) \rightarrow S'$$

3. Otro componente de un problema es la **función de prueba de objetivo o test objetivo**. La función test objetivo permite verificar si el estado actual del agente corresponde al objetivo que se quiere lograr y que representa la solución al problema. La función test objetivo tiene como entrada un estado y retorna un valor de verdadero o falso, verdadero en el caso de que se haya alcanzado el objetivo y falso en caso contrario.

$$\text{FuncionObjetivo}(S) \rightarrow \begin{cases} \text{Verdadero,} & \text{se alcanzo el objetivo} \\ \text{falso,} & \text{no se alcanzo el objetivo} \end{cases}$$

4. El cuarto componente es la **función de costo**. La función de costo toma como entrada una ruta (conjunto de acciones de transiciones de estados tomadas por el agente) arroja la penalidad o costo de haber tomado la secuencia de acciones teniendo como entrada la ruta tomada y retorna un número el cual es el costo o penalidad para el agente de haber tomado ese conjunto de acciones de transición de estados.

$$\text{Funcion de Costo}(S^a \rightarrow S^a \rightarrow S) = n$$

Generalmente la función de costo es una función aditiva, es decir que el costo de tomar una ruta viene dado por la sumatoria de la función de costo tomada en cada acción que le llevó a un estado particular. Cada acción tiene entonces un costo individual asociado y se puede definir como una función cuya entrada es un estado, una acción y el estado resultante de tomar dicha acción.

$$\text{Costo de una Acción}(S, a, S') = n$$

3.2. Resolución de problemas de búsqueda de ruta

El problema de búsqueda de ruta es un típico problema de navegación en el cual se debe conseguir la ruta de un punto *A* a un punto *B* que cumpla con ciertos requisitos. Un ejemplo claro, tenemos el mapa de un país (ver Figura 3), el cual no conocemos y queremos ir de una ciudad a otra. El problema que se nos plantea en este caso es partir desde la ciudad de Arad y encontrar la mejor ruta para llegar a la capital de Rumanía, Bucarest. Ahora bien, dado el mapa completo de rutas de Rumanía y partiendo desde Arad, un agente inteligente debe definir una secuencia de acciones que nos llevará a describir la ruta a seguir para llegar al destino, en nuestro caso, el destino sería Bucarest. Si observamos las distintas rutas en el mapa podemos ver que existen varias opciones de rutas desde Arad hasta Bucarest (sin tomar en cuenta otros factores externos como el tráfico, clima, etc.). Una ruta puede ser, por ejemplo: Arad-Sibiu-Fagaras-Bucarest, que puede contar como una posible solución al problema inicial. Se describen una secuencia de pasos y acciones que combinadas nos da la garantía de lograr nuestro objetivo, llegar a la ciudad de Bucarest, partiendo desde Arad.

En la Figura 4 tenemos un mapa simplificado completo de las rutas de Rumanía, que nos llevarán desde Arad hasta Bucarest. El problema ahora radica en encontrar la ruta correcta (preferiblemente la más corta) a la ciudad de Bucarest, partiendo desde Arad. Para el ejemplo se tiene en cuenta otro factor, el costo asociado de ir de una ciudad a otra. Por ejemplo, ir de Arad a Zerind tiene un costo de 75, de Arad a Oradea, solo hay una forma de llegar ejecutando dos acciones desde Arad a Oradea pasando por Zerind y el costo asociado sería el costo asociado a la ruta entre Arad y Oradea, el cual sería $75+71 = 146$. Dicho costo en este caso corresponde al número de kilómetros que separa ambas ciudades.

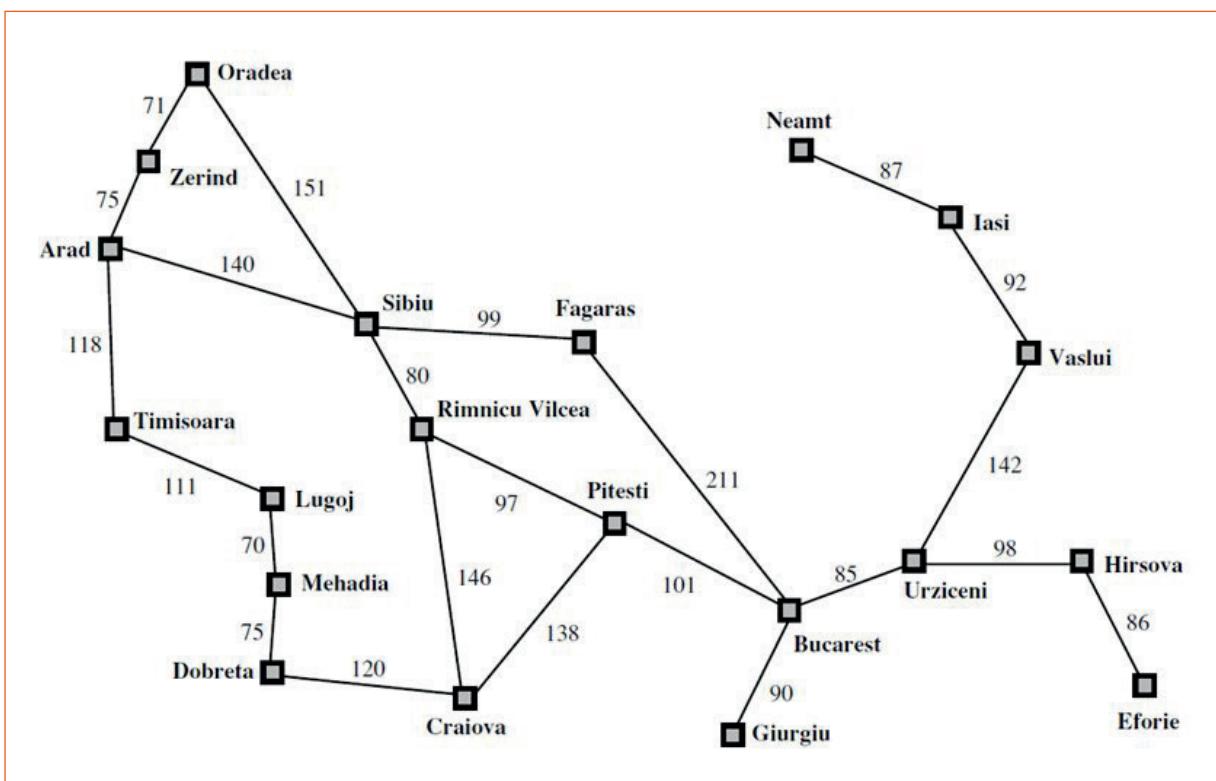


Figura 4. Mapa simplificado de parte de Rumanía. Recuperado de Russell y Norvig (2010).

1. **El estado inicial:** sería la ciudad de Arad.
2. **Las acciones:** en nuestro ejemplo de Rumanía, al estar en una ciudad, disponemos de diferentes rutas que nos llevan a ciudades vecinas, pero no podemos ir a todas las ciudades desde una ciudad cualquiera.

Desde Arad por ejemplo podemos solo ir a Zerind, Timisoara o Sibiu. Desde Zerind solo a Oradea o regresarnos a Arad. Y así sucesivamente.

$$\text{Acción ('en Arad')} \rightarrow \{'\text{ir a Zerind}', '\text{ir a Timisoara}'\}$$

$$\text{Acción ('en Zerind')} \rightarrow \{'\text{ir a Arad}', '\text{ir a Oradea}'\}$$

Cada acción tiene un resultado, estando 'en Arad' de la acción 'ir a Zerind' sería:

$$\text{Resultado ('en Arad', 'ir a Zerind')} \rightarrow \text{'en Zerind'}$$

Si el agente está en la ciudad de Arad y toma la acción de dirigirse hacia la ciudad de Timisoara, el resultado de tomar dicha acción generaría un nuevo estado al agente de estar en la ciudad de Timisoara.

$$\text{Resultado ('en Arad', 'ir a Timisoara')} \rightarrow \text{'en Timisoara'}$$

3. En este caso el **objetivo** sería estar en la ciudad de destino, la ciudad de Bucarest. Así que para el estado 'en Bucharest' del agente, la función objetivo sería verdadero y para el estado de las demás ciudades de Rumanía sería Falso.

$$\text{FuncionObjetivo}(S) \rightarrow \begin{cases} \text{Verdadero,} & \text{si } S = \text{'en Bucarest'} \\ \text{falso,} & \text{si } S \neq \text{'en Bucarest'} \end{cases}$$

En el ejemplo de la búsqueda de una ruta hacia la ciudad de Bucarest, el costo asociado a cada acción puede ser la distancia o el tiempo de ir de una ciudad a otra. En la imagen ilustrada se le asoció un costo a cada acción, la distancia.

4. El otro componente sería la **función de costo**. La función de costo toma como entrada un camino (conjunto de acciones de transiciones de estados tomadas por el agente), arroja la penalidad o costo de haber tomado la secuencia de acciones teniendo como entrada el camino tomado y retorna un número que es el costo o penalidad para el agente de haber tomado ese conjunto de acciones de transición de estados.

$$\text{Funcion de Costo }(S^a \rightarrow S^a) = n$$

Generalmente la función de costo es una función aditiva, es decir que el costo de una ruta viene dado por la sumatoria de la función de costo tomada en cada acción que le llevó a un estado particular. Cada acción tiene entonces un costo asociado y se puede definir como una función cuya entrada es un estado, una acción y el estado resultante de tomar dicha acción.

$$\text{Costo de una Acción }(S, a, S') = n$$

3.3. Soluciones de árboles de búsqueda

Estos problemas de búsqueda de ruta pueden ser manejados como los problemas de árboles de búsqueda. Por ejemplo: podemos dividir en tres partes el mapa simplificado de estados de Rumanía (ver la Figura 4):

1. Estados explorados (serían los estados que ya son parte de los caminos explorados).
2. Estados bordes (son los estados visitados, pero en los que no hemos explorado sus acciones posibles).
3. Estados inexplorados (aún no han sido visitados en ninguna de las acciones tomadas).

Así, por ejemplo, el siguiente mapa contendría los siguientes estados explorados, bordes e inexplorados (ver Figura 5):

```

Explorados→{'en Arad', 'en Zerind', 'en Timisioara',
            'en Sibiu','en Rimnicu Vilcea'}
Bordes→{'en Oradea', 'en Fagaras', 'en Pitesti', 'en Lugoj'}
Inexplorados→{...el resto}

```

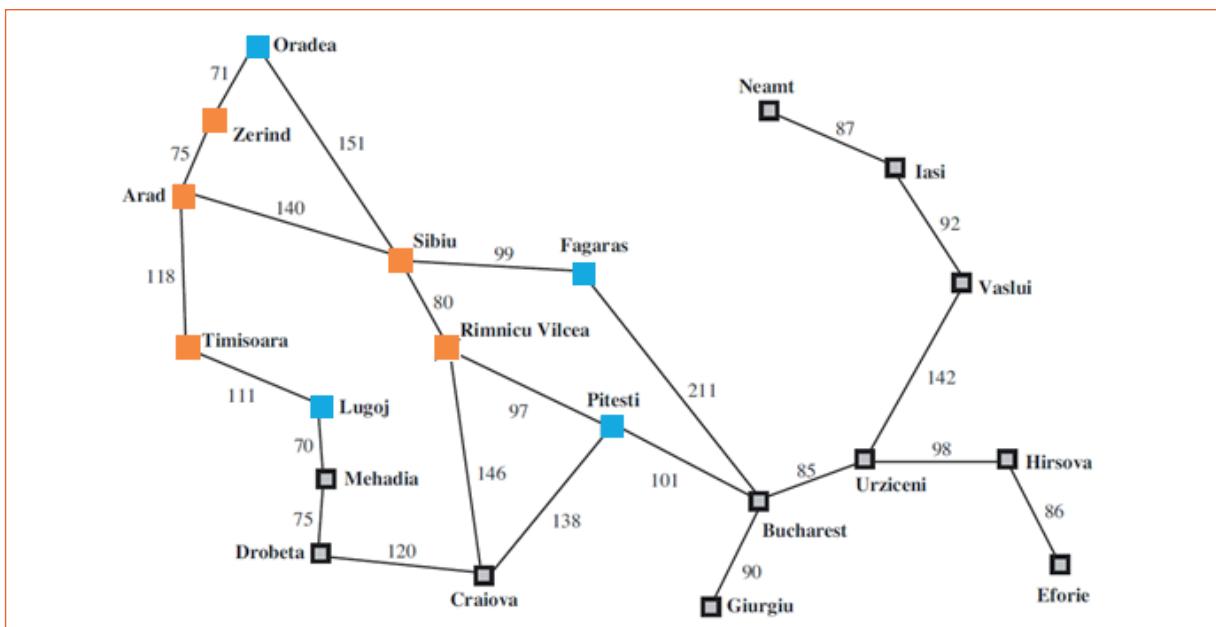


Figura 5. Mapa simplificado de parte de Rumanía. Adaptado de Russell y Norvig (2010).

Algoritmo de árboles de búsqueda:

```

def arboldebusqueda (problema):
    frontera = [Estado Inicial]
    Loop:
        if frontera is empty: return 'Fail'
        path = remove_choice(frontera)

```

```

s = path.end()
if s is the_goal: return path
for a in actions:
    add [path + a->Result(s,a)] to frontera
    
```

El algoritmo de árboles de búsqueda crea el conjunto frontera inicialmente, con el estado inicial al problema. Entra en un ciclo hasta encontrar el objetivo o simplemente no se encuentra una solución, lo cual puede ser posible, si la ciudad no se encuentra dentro del mapa de Rumanía, por ejemplo. Entonces se verifica si el conjunto frontera está vacío. Si está vacío es porque se recorrió todo el árbol y no se logró llegar al estado objetivo. Si no está vacío, se toma uno de los estados de la frontera de la ruta. Si el final de la ruta es el estado objetivo, hemos completado la búsqueda y hemos encontrado una ruta al objetivo. Si no, entonces se toma del conjunto de acciones y agregamos la acción a la ruta que nos lleva al estado que se tomó de la frontera y agregamos este nuevo estado resultado al conjunto fronteras. Así tenemos una nueva ruta construida con la nueva acción y el resultado de aplicar la acción al estado. El corazón del método de búsqueda radica en cómo escogemos el próximo estado y la acción que nos lleva a ese estado (*función remove_choice(frontera)*). Para ello existen varios algoritmos.

3.4. Estrategias de búsqueda no informada

En esta sección se expondrán las estrategias de búsqueda en estructuras de grafos y árbol, en la que no se tiene información adicional a la que define el problema. En algunos casos de búsqueda en recorrido de grafos es necesario tener información adicional para acelerar la búsqueda. Existen otras estrategias de búsqueda llamadas **búsqueda informada o heurísticas**, en las que se tienen mayor información acerca de cómo alcanzar el nodo objetivo de manera más rápida según el planteamiento del problema. Pero esto se tratará más adelante.

3.4.1. Búsqueda primero en anchura

El método de búsqueda primero en anchura **siempre selecciona de la frontera uno de los que no habían sido considerados todavía y que sea el más corto posible**. En el ejemplo de llegar a Bucarest, la frontera inicialmente sería conformada solo por la ciudad de Arad.

$$\text{frontera} = \{ \text{'en Arad}' \}$$

Se toma el único elemento de la frontera y se elimina de fronteras; se expanden todas las posibles acciones que se pueden tomar desde el estado inicial, en este caso, y los resultados de dichas acciones. Obteniendo en este caso tres rutas posibles.

$$\text{path}_1 = \{ \text{'Arad'} \rightarrow \text{'Sibiu'} \}$$

$$\text{path}_2 = \{ \text{'Arad'} \rightarrow \text{'Zerind'} \}$$

$$\text{path}_3 = \{ \text{'Arad'} \rightarrow \text{'Timisoara'} \}$$

Se hace la siguiente pregunta: ¿El estado final de cualquiera de estas rutas dadas por el conjunto *path* es el estado objetivo? No, entonces agregamos estos estados finales al conjunto frontera y volvemos al ciclo, quedando frontera de la siguiente manera:

$$\text{frontera} = \{ \text{'en Zerind'}, \text{'en Sibiu'}, \text{'en Timisoara'} \}$$

En el próximo paso se debe seleccionar entonces el camino o ruta más corta, según este algoritmo, pero en este caso la longitud de las diferentes rutas es 1, así que en teoría se podría escoger cualquiera. Sea tomado el *path*, Ahora se elimina de frontera 'en Sibiu' y se expanden las rutas desde este estado. Desde Sibiu se tiene entonces cuatro acciones que llevarían a los estados de Arad, Fagaras, Oradea y Rimnica Vilcea. Si se observa la acción que lleva la ruta de nuevo al estado de Arad, es una de las acciones posibles y se puede caer en un ciclo infinito. Es aquí donde es importante llevar el registro de los estados explorados, para evitar regresar a estados que ya se habían visitado. Para ello en vez de tratar el problema como una estructura de árbol, tratamos el problema como una estructura de grafos. Al ser una estructura de árbol se puede agregar a cualquier nodo del árbol un estado que ya haya sido visitado y se caería en un ciclo ineficiente, justamente porque la búsqueda en estructuras de árbol se mueve de nodo padre a nodo hijo (de arriba hacia abajo) y no de nodo hijo a nodo padre, por lo tanto, se podrían crear ciclos infinitos en las rutas. Mientras que, en una estructura de grafos, podemos llevar el registro de los nodos explorados para no volverlos a marcar como posibles estados en las rutas. Quedando el algoritmo de búsqueda modificado de la siguiente manera:

```
def busquedaenGrafos (problema):
    frontera = [Estado Inicial]
    explorados = []
    Loop:
        if frontera is empty: return 'Fail'
        path = remove_choice(frontera)
        s = path.end()
        add s to explorados
        if s is the_goal: return path
        for a in actions:
            add [path + a → Result(s,a)] to frontera
            al menos que Result(s, a) esté en fronteras o explorados
```

De esta manera cuando se expanden las rutas desde Arad, se agrega 'en Arad' a explorados.

$$\text{explorados} = \{ \text{'en Arad'} \}$$

Así cuando se tome 'en Sibiu' en la próxima iteración, tendríamos las rutas:

$$\text{path}_1 = \{ \text{'Arad'} \rightarrow \text{'Sibiu'} \rightarrow \text{'Fagaras'} \}$$

$$\text{path}_4 = \{ \text{'Arad'} \rightarrow \text{'Sibiu'} \rightarrow \text{'Oradea'} \}$$

$$\text{path}_5 = \{ \text{'Arad'} \rightarrow \text{'Sibiu'} \rightarrow \text{'Rimnicu Vilcea'} \}$$

$$path_2 = \{ 'Arad' \rightarrow 'Zerind' \}$$

$$path_3 = \{ 'Arad' \rightarrow 'Timisoara' \}$$

Se tienen entonces las correspondientes rutas a las dos que quedaron de la iteración anterior y se expande la ruta desde '**Sibiu**'. Se agrega el estado '**en Sibiu**' a explorados.

$$explorados = \{ 'en Arad' , 'en Sibiu' \}$$

Se expande '**en Sibiu**' y se agregan a fronteras los estados que se obtienen de dicha expansión.

$$frontera = \{ 'en Zerind' , 'en Timisoara' , 'en Oradea' , 'en Fagaras' , 'en Rimnicu Vilcea' \}$$

En la siguiente iteración digamos que se toma el $path_2$ por ser una de las rutas más cortas. Se elimina '**en Zerind**' de frontera, se agrega a explorados y se expande, pero la única ciudad posible es '**en Oradea**' y ya está en frontera, así que en esta iteración no hay nada que agregar a frontera. La siguiente ruta más corta posible es el $path_3 = \{ 'Arad' \rightarrow 'Timisoara' \}$. Se elimina '**en Timisoara**' de frontera. Se agrega una nueva ruta quedando $path_3 = \{ 'Arad' \rightarrow 'Timisoara' \rightarrow 'Lugoj' \}$. Se agrega '**en Timisoara**' a explorados y '**en Lugoj**' a frontera.

Ahora tenemos:

$$path_1 = \{ 'Arad' \rightarrow 'Sibiu' \rightarrow 'Fagaras' \}$$

$$path_4 = \{ 'Arad' \rightarrow 'Sibiu' \rightarrow 'Oradea' \}$$

$$path_5 = \{ 'Arad' \rightarrow 'Sibiu' \rightarrow 'Rimnicu Vilcea' \}$$

$$path_2 = \{ 'Arad' \rightarrow 'Zerind' \rightarrow 'Oradea' \}$$

$$path_3 = \{ 'Arad' \rightarrow 'Timisoara' \rightarrow 'Lugoj' \}$$

y,

$$explorados = \{ 'en Arad' , 'en Sibiu' , 'en Timisoara' , 'en Zerind' \}$$

$$frontera = \{ 'en Fagaras' , 'en Lugoj' , 'en Oradea' , 'en Rimnicu Vilcea' \}$$

Ahora todas las rutas son de longitud 2, se podría escoger cualquiera de ellas. Supongamos que se escoge el $path_1$. Se elimina '**en Fagaras**' de frontera. Se expande '**Fagaras**', la única ciudad nueva sería Bucarest. Se agrega '**en Bucarest**' al $path_1 = \{ 'Arad' \rightarrow 'Sibiu' \rightarrow 'Fagaras' \rightarrow 'Bucarest' \}$ y a frontera. Sin embargo, en este punto el algoritmo aún no termina, porque el objetivo se verifica una vez que se haya eliminado dicho estado de frontera, en este momento se está agregando. Además, porque el algoritmo termina cuando consigue el camino más cercano al destino. A pesar de que con la ruta $path_1$, en la iteración se haya alcanzado la ciudad objetivo, primero no se ha verificado y segundo no se han verificado las demás rutas, para determinar si es la ruta más corta.

Se podría hacer un corte temprano al algoritmo al encontrar la ciudad objetivo, sin embargo, no se puede saber si realmente fue la ruta más corta si no se verifican las demás rutas, o si existe más de una ruta corta de la misma longitud. El algoritmo se puede optimizar si al agregar a la frontera se verifica si

se llegó al punto objetivo o podríamos agregar la función de costo distancia y verificar si realmente no existe otra ruta con distancia más corta a la ya encontrada. Lo importante con este método es que el algoritmo de Búsqueda primero en anchura garantiza el hecho de que se encontrará la ruta más corta en términos de pasos en el grafo. Sin embargo, si se está buscando la ruta más corta en términos de distancia, se debe tomar en cuenta entonces la función de costo como la distancia y validar el costo total de tomar cada ruta.

3.4.2. Búsqueda primero en anchura de costo uniforme

Este método de búsqueda de costo uniforme **selecciona el estado o nodo que se va a expandir según su costo y expandirá el de menor costo acumulado**. En cuyo caso si todos los costos son iguales para la expansión de cada nodo, este método funcionaría igual que el de búsqueda primero en anchura. El método de búsqueda de costo uniforme no toma en cuenta la longitud de las rutas para seleccionar la más corta sino evalúa el costo acumulado en cada ruta y escoge la que tenga el menor costo. Algoritmo también llamado **algoritmo de búsqueda de anchura de menor costo**.

Del ejemplo anterior, paso inicial:

$$\text{frontera} = \{ \text{'en Arad'} \}$$

Se elimina '*en Arad*' de frontera, se expande '*en Arad*' y se agregan las rutas:

$$\text{path}_1 = \{ \text{'Arad'} \xrightarrow{140} \text{'Sibiu'} \}$$

$$\text{path}_2 = \{ \text{'Arad'} \xrightarrow{75} \text{'Zerind'} \}$$

$$\text{path}_3 = \{ \text{'Arad'} \xrightarrow{118} \text{'Timisoara'} \}$$

Se agregan a frontera:

$$\text{frontera} = \{ \text{'en Sibiu'}, \text{'en Zerind'}, \text{'en Timisoara'} \}$$

y a explorados:

$$\text{explorados} = \{ \text{'en Arad'} \}$$

En la siguiente iteración la ruta a escoger es la de menor costo, en este caso sería el path_2 .

Se elimina '*en Zerind*' de frontera, se expande y se agregan la nueva ruta:

$$\text{path}_2 = \{ \text{'Arad'} \xrightarrow{75} \text{'Zerind'} \xrightarrow{71} \text{'Oradea'} \}, \text{costo total} = 146$$

Se agregan a frontera:

$$\text{frontera} = \{ \text{'en Sibiu'}, \text{'en Oradea'}, \text{'en Timisoara'} \}$$

y a explorados:

$$\text{explorados} = \{ \text{'en Arad'}, \text{'en Zerind'} \}$$

Ahora tenemos tres rutas path_1 con costo 140, path_2 con costo 146 y path_3 con costo 118. El próximo nodo a expandirse ahora sería el nodo '*en Timisoara*' de la ruta path_3 , dado que es la ruta con menor costo (118).

Se elimina '*en Timisoara*' de frontera, se expande y se agregan a las rutas nuevas, quedando:

$$\text{path}_3 = \{ \text{'Arad'} \xrightarrow{118} \text{'Timisoara'} \xrightarrow{111} \text{'Lugoj'} \}, \text{costo total} = 229$$

Se agregan a frontera:

$$\text{frontera} = \{ \text{'en Sibiu'}, \text{'en Oradea'}, \text{'en Lugoj'} \}$$

y a explorados:

$$\text{explorados} = \{ \text{'en Arad'}, \text{'en Zerind'}, \text{'en Timisoara'} \}$$

Aún no se consigue un nodo al final de cada ruta la ciudad de Bucarest así que se sigue iterando. En la próxima iteración, la ruta menos costosa sería path_1 . Con la ciudad de Sibiu. Se elimina '*en Sibiu*' de frontera, se expande y se agregan las nuevas rutas:

$$\text{path}_1 = \{ \text{'Arad'} \xrightarrow{140} \text{'Sibiu'} \xrightarrow{99} \text{'Fagaras'} \}, \text{costo total} = 239$$

$$\text{path}_4 = \{ \text{'Arad'} \xrightarrow{140} \text{'Sibiu'} \xrightarrow{80} \text{'Rimnicu Vilcea'} \}, \text{costo total} = 220$$

No se agrega la ruta $\text{path}_4 = \{ \text{'Arad'} \xrightarrow{140} \text{'Sibiu'} \xrightarrow{80} \text{'Oradea'} \}$, porque Oradea ya está en frontera. Dicha ruta queda descartada y se elimina.

Se agregan a frontera:

$$\text{frontera} = \{ \text{'en Lugoj'}, \text{'en Fagaras'}, \text{'en Rimnicu Vilcea'} \}$$

y a explorados:

$$\text{explorados} = \{ \text{'en Arad'}, \text{'en Zerind'}, \text{'en Timisoara'}, \text{'en Sibiu'}, \text{'en Oradea'} \}$$

En la siguiente iteración el camino más barato sería el path_2 con un costo de 146. Así que se agrega a explorados, pero no hay nada que agregar a las rutas ni a fronteras, porque al expandir el nodo de Oradea, se tiene ya '*en Fagaras*' en frontera, y dicha ruta ya no sigue tomándose en cuenta. Así que no se agrega ninguna ruta.

Se toma la próxima ruta más corta, que sería path_4 . Así que se agrega a explorados '*en Rimnicu Vilcea*', se elimina de frontera y se agregan las rutas:

$\text{path}_4 = \{ \text{'Arad'} \xrightarrow{140} \text{'Sibiu'} \xrightarrow{80} \text{'Rimnicu Vilcea'} \xrightarrow{97} \text{'Pitesti'} \}, \text{costo total} = 317$

$\text{path}_5 = \{ \text{'Arad'} \xrightarrow{140} \text{'Sibiu'} \xrightarrow{80} \text{'Rimnicu Vilcea'} \xrightarrow{146} \text{'Craiova'} \}, \text{costo total} = 366$

Se agregan a frontera:

$\text{frontera} = \{ \text{'en Lugo}', \text{'en Fagaras'}, \text{'Pitesti'}, \text{'Craiova'} \}$

y a explorados:

$\text{explorados} = \{ \text{'en Arad'}, \text{'en Zerind'}, \text{'en Timisoara'}, \text{'en Sibiu'}, \text{'en Oradea'}, \text{'en Rimnicu Vilcea'} \}$

Aun no se alcanza el objetivo, se toma la próxima ruta más corta, que sería path_3 . Así que se agrega a explorados 'en Lugo', se elimina de frontera y se agregan las rutas:

$\text{path}_3 = \{ \text{'Arad'} \xrightarrow{118} \text{'Timisoara'} \xrightarrow{111} \text{'Lugo'} \xrightarrow{70} \text{'Mehadia'} \}, \text{costo total} = 299$

Se agregan a frontera:

$\text{frontera} = \{ \text{'en Fagaras'}, \text{'Pitesti'}, \text{'Craiova'}, \text{'en Mehadia'} \}$

y a explorados:

$\text{explorados} = \{ \text{'en Arad'}, \text{'en Zerind'}, \text{'en Timisoara'}, \text{'en Sibiu'}, \text{'en Oradea'}, \text{'en Rimnicu Vilcea'}, \text{'en Lugo'} \}$

La próxima ruta más corta sería path_1 . Se agrega a explorados 'en Fagaras', se elimina de frontera y se agregan las rutas:

$\text{path}_1 = \{ \text{'Arad'} \xrightarrow{140} \text{'Sibiu'} \xrightarrow{99} \text{'Fagaras'} \xrightarrow{211} \text{'Bucarest'} \}, \text{costo total} = 450$

Se agregan a frontera:

$\text{frontera} = \{ \text{'Pitesti'}, \text{'Craiova'}, \text{'en Mehadia'}, \text{'en Bucarest'} \}$

y a explorados:

$\text{explorados} = \{ \text{'en Arad'}, \text{'en Zerind'}, \text{'en Timisoara'}, \text{'en Sibiu'}, \text{'en Oradea'}, \text{'en Rimnicu Vilcea'}, \text{'en Lugo'}, \text{'en Bucarest'} \}$

En este punto, aunque en una de las rutas se alcanzó la ciudad objetivo aún no se sabe cuál de las rutas es la menos costosa, por lo que se debe seguir iterando. Así que en la próxima iteración el próximo nodo candidato sería el de la ruta path_3 . Quedando entonces:

$\text{path}_3 = \{ \text{'Arad'} \xrightarrow{118} \text{'Timisoara'} \xrightarrow{111} \text{'Lugo'} \xrightarrow{70} \text{'Mehadia'} \xrightarrow{70} \text{'Drobeta'} \}, \text{costo total} = 369$

Se agregan a frontera:

$$\text{frontera} = \{ \text{'Pitesti'}, \text{'Craiova'}, \text{'en Bucarest'}, \text{'Drobeta'} \}$$

y a explorados:

$$\text{explorados} = \{ \text{'en Arad'}, \text{'en Zerind'}, \text{'en Timisoara'}, \text{'en Sibiu'}, \text{'en Oradea'}, \text{'en Rimnicu Vilcea'}, \text{'en Lugoj'}, \text{'en Bucarest'}, \text{'Mehadia'} \}$$

La próxima iteración el nodo candidato sería el de la ruta $\text{path}_4 =$. Quedando entonces:

$$\text{path}_4 = \{ \text{'Arad'} \xrightarrow{140} \text{'Sibiu'} \xrightarrow{80} \text{'Rimnicu Vilcea'} \xrightarrow{97} \text{'Pitesti'} \xrightarrow{101} \text{'Bucarest'} \}, \text{costo total} = 418$$

$$\text{path}_6 = \{ \text{'Arad'} \xrightarrow{140} \text{'Sibiu'} \xrightarrow{80} \text{'Rimnicu Vilcea'} \xrightarrow{97} \text{'Pitesti'} \xrightarrow{138} \text{'Craiova'} \}, \text{costo total} = 546$$

Sin embargo, tiene peor costo que la ruta del menor costo así que se elimina.

Se agregan a frontera:

$$\text{frontera} = \{ \text{'Craiova'}, \text{'en Bucarest'}, \text{'Drobeta'} \}$$

y a explorados:

$$\text{explorados} = \{ \text{'en Arad'}, \text{'en Zerind'}, \text{'en Timisoara'}, \text{'en Sibiu'}, \text{'en Oradea'}, \text{'en Rimnicu Vilcea'}, \text{'en Lugoj'}, \text{'en Bucarest'}, \text{'en Mehadia'}, \text{'en Pitesti'} \}$$

Ahora se tiene una ruta con el nodo objetivo y menos costoso que la primera ruta encontrada, sin embargo, aún quedan otras rutas por explorar. La próxima iteración correspondería a $\text{path}_5 =$. Quedando:

$$\text{path}_5 = \{ \text{'Arad'} \xrightarrow{140} \text{'Sibiu'} \xrightarrow{80} \text{'Rimnicu Vilcea'} \xrightarrow{146} \text{'Craiova'} \xrightarrow{120} \text{'Drobeta'} \}, \text{costo total} = 486, \text{es mayor que la ruta de menor costo, así que se elimina también.}$$

Se agregan a frontera:

$$\text{frontera} = \{ \text{'en Bucarest'}, \text{'Drobeta'} \}$$

y a explorados:

$$\text{explorados} = \{ \text{'en Arad'}, \text{'en Zerind'}, \text{'en Timisoara'}, \text{'en Sibiu'}, \text{'en Oradea'}, \text{'en Rimnicu Vilcea'}, \text{'en Lugoj'}, \text{'en Bucarest'}, \text{'en Mehadia'}, \text{'en Craiova'} \}$$

La próxima ruta sería $\text{path}_3 =$. La cual nos llevaría a tener:

$$\text{path}_1 = \{ \text{'Arad'} \xrightarrow{140} \text{'Sibiu'} \xrightarrow{99} \text{'Fagaras'} \xrightarrow{211} \text{'Bucarest'} \}, \text{costo total} = 450$$

$$\text{path}_4 = \{ \text{'Arad'} \xrightarrow{140} \text{'Sibiu'} \xrightarrow{80} \text{'Rimnicu Vilcea'} \xrightarrow{97} \text{'Pitesti'} \xrightarrow{101} \text{'Bucarest'} \}, \text{costo total} = 418$$

$\text{path}_3 = \{ \text{'Arad'} \xrightarrow{118} \text{'Timisoara'} \xrightarrow{111} \text{'Lugoj'} \xrightarrow{70} \text{'Mehadia'} \xrightarrow{120} \text{'Drobeta'} \rightarrow \text{'Craiova'} \}$, costo total = 489, nuevamente otra ruta con costo mayor a la ruta menos costosa es descartada.

frontera queda:

$$\text{frontera} = \{ \text{'en Bucarest'} \}$$

y explorados:

$\text{explorados} = \{ \text{'en Arad}', \text{'en Zerind}', \text{'en Timisoara}', \text{'en Sibiu}', \text{'en Oradea}', \text{'en Rimnicu Vilcea'}, \text{'en Lugoj'}, \text{'en Bucarest'}, \text{'en Mehadia'}, \text{'en Drobeta'}, \text{'en Craiova'} \}$

Quedando la ruta menos costosa la del path_4 .

$\text{path}_4 = \{ \text{'Arad'} \xrightarrow{140} \text{'Sibiu'} \xrightarrow{80} \text{'Rimnicu Vilcea'} \xrightarrow{97} \text{'Pitesti'} \xrightarrow{101} \text{'Bucarest'} \}$, costo total = 418

Aun cuando se haya encontrado el nodo objetivo el algoritmo no termina hasta encontrar la ruta más corta al objetivo. Esto asegura al algoritmo alcanzar la ruta menos costosa al nodo objetivo.

3.4.3. Búsqueda primero en profundidad

La búsqueda primero en profundidad **siempre va expandiendo los nodos que sean más profundos, así que no es un método óptimo para encontrar el camino más corto o el menos costoso. Es común usar este método de manera recursiva.**

En comparación con los otros métodos, el método de búsqueda en profundidad no es precisamente el mejor (ver Figura 6).

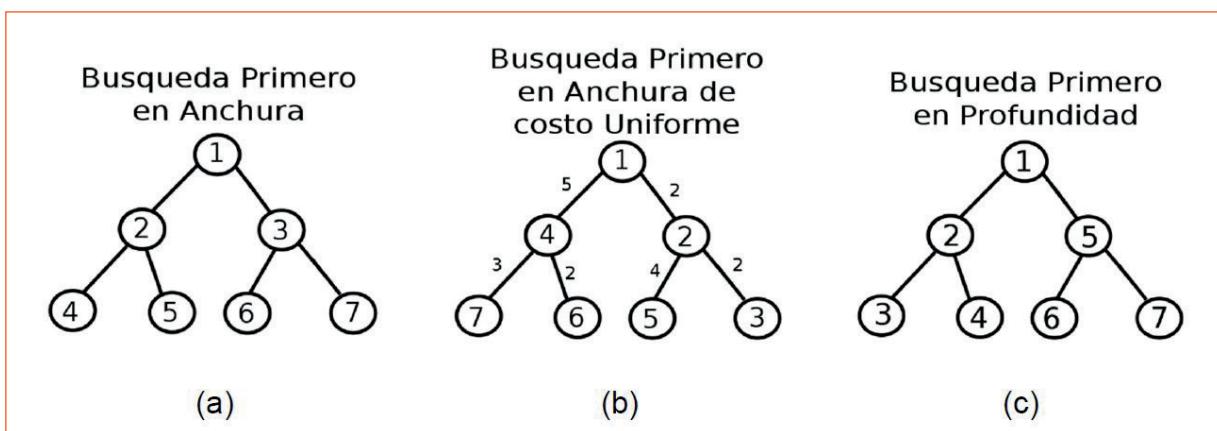


Figura 6. Ejemplo ilustrado que muestra el recorrido en una estructura de árbol la búsqueda primero en anchura (a), primero en anchura de costo uniforme (b) y primero en profundidad (c). Los números en los nodos indican el orden del recorrido.

El método de primero en profundidad puede no ser óptimo, sin embargo, es bastante eficiente en espacio de almacenamiento requerido. Cuando se trabaja en árbol de búsqueda binario cuyo tamaño tiende al infinito, la capacidad de almacenamiento que se requiere para el algoritmo de primero en profundidad es mucho menor que los otros dos métodos.

Supongamos que tenemos un árbol binario cuya profundidad tiende al infinito, el recorrido es más o menos como sigue, siguiendo la forma de la mancha en la Figura 7.

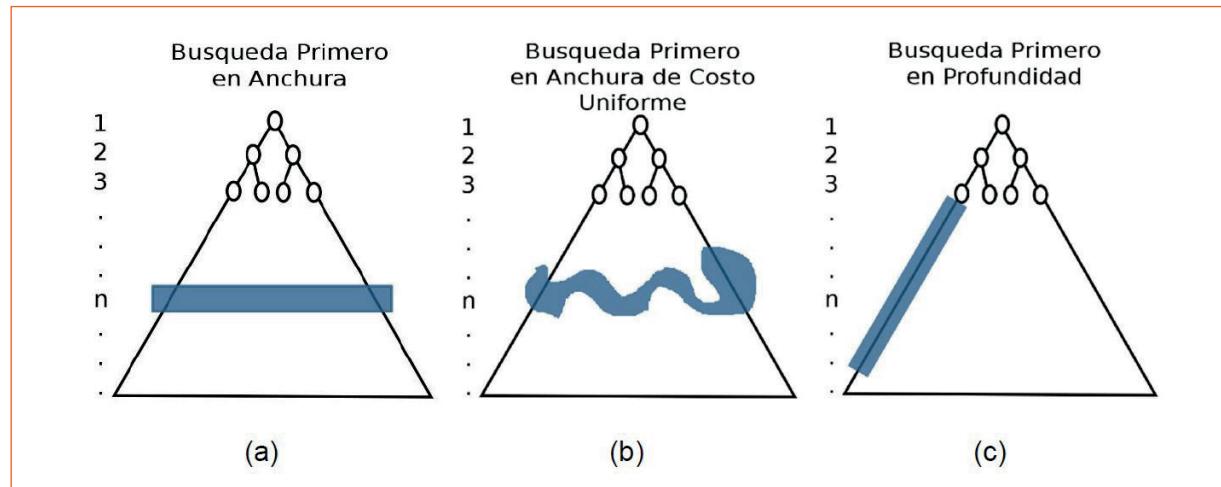


Figura 7. Ejemplo ilustrado que muestra el recorrido en una estructura de árbol la búsqueda primero en anchura (a), primero en anchura de costo uniforme (b) y primero en profundidad (c) cuando la altura del árbol tiende a infinito.

La capacidad de almacenamiento en el algoritmo de búsqueda en profundidad es menor que en los de búsqueda primero en anchura o el de costo uniforme. Otra de las diferencias entre estos algoritmos es la completitud, si realmente se consigue el nodo objetivo en la búsqueda, cuando la profundidad del árbol tiende a infinito. Para el caso de búsqueda primero en profundidad o de costo uniforme, definitivamente, el algoritmo garantiza que el nodo objetivo será alcanzado, sin embargo, no sucede lo mismo con el método de búsqueda primero en profundidad, dado que el algoritmo busca el nodo más profundo antes de ir a otra raíz del árbol, y si la profundidad tiende a infinito, jamás se conseguirá el nodo más profundo.

Dentro de los algoritmos de búsqueda primero en profundidad hay algunas variantes:

- Búsqueda de profundidad limitada:** este método resolvería el hecho de caer en árboles infinitos, acotando la búsqueda a un nivel *l* de profundidad del árbol. Sin embargo, esto no garantiza la completitud del algoritmo si el punto objetivo se encuentra en niveles superiores al árbol. Otra limitación pudiera ser la distancia, sin embargo, se tendría que conocer a priori la solución al problema para saber determinar mejores los valores límites.
- Búsqueda primero en profundidad con profundidad iterativa:** este método se usa generalmente en combinación con el método de búsqueda primero en profundidad, estableciendo un límite inicial y si aún no se encuentra el nodo objetivo ir incrementando gradualmente el límite de profundidad (primero en 1, luego en 2, y así sucesivamente). Hasta que encontramos el nodo objetivo. Por lo general este método es utilizado cuando hay un espacio grande de búsqueda y no se conoce la profundidad de la solución.

3.5. Estrategias de búsqueda informada

Las estrategias de búsqueda informada son estrategias de búsqueda heurísticas, en las cuales además de la información referente al problema de conseguir un punto objetivo en un espacio definido por

un grafo, se tiene información adicional que ayudaría o facilitaría la búsqueda. Veamos en la Figura 8 un ejemplo de cómo funciona el método de búsqueda de costo uniforme:

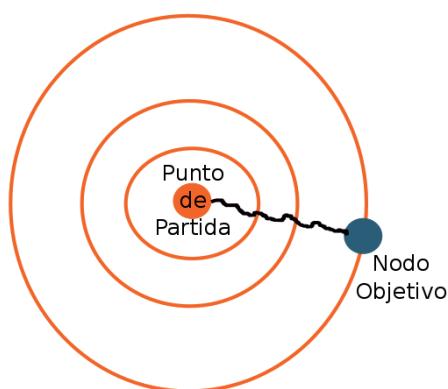


Figura 8. A modo de ilustración se muestra el alcance del recorrido de un algoritmo de búsqueda de costo uniforme.

3.5.1. Búsqueda codiciosa primero el mejor

En la Figura 8, tenemos un ejemplo de cómo funciona el algoritmo de búsqueda de costo uniforme a partir de un estado inicial (punto de partida). El algoritmo evalúa la región cerca de su entorno y a medida que va consiguiendo la ruta menos costosa se sigue moviendo hacia la parte más externa, realizando una búsqueda uniforme en el área alrededor del punto de partida, hasta conseguir el objetivo. En caso de que el espacio de búsqueda sea pequeño, no habría mayores problemas, pero en un espacio de búsqueda más grande sería demasiado ineficiente. En todo caso si no se tiene información adicional más que conocer cuál es el punto de partida y objetivo, no hay nada que se pueda hacer, para mejorar el algoritmo. Si queremos mejorar el algoritmo a fin de que pueda encontrar de manera más rápida el objetivo, tendríamos que tener más conocimiento acerca del problema, el tipo de conocimiento que ha sido probado ser el más útil en problemas de búsqueda es la distancia aproximada del punto de partida al objetivo. Conociendo a priori la distancia entre estos dos puntos, podemos limitar la búsqueda en la dirección que nos acerque más al objetivo (ver Figura 9).

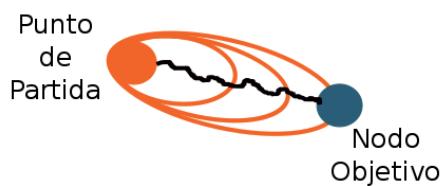


Figura 9. A modo de ilustración se muestra recorrido de un algoritmo con estrategia de búsqueda informada, llamado búsqueda codiciosa.

Este algoritmo es llamado el algoritmo de búsqueda codicioso. Este algoritmo busca expandir la búsqueda en aquellos nodos cuya distancia al objetivo sea la más cercana. Así en vez de explorar en círculos alrededor del punto de partida, se reduce el área de búsqueda a aquellos nodos que más se acercan al objetivo, de tal manera que la búsqueda nos lleva directamente al objetivo. Sin embargo,

no siempre es el caso, ¿qué pasa si entre el espacio de búsqueda encontramos algún obstáculo, como en el ejemplo ilustrado de la Figura 10?

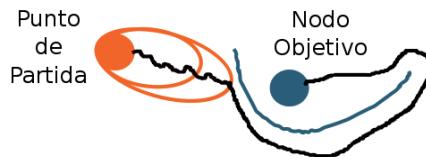


Figura 10. A modo de ilustración se muestra recorrido de un algoritmo con estrategia de búsqueda codicosa, cuando se le presenta un obstáculo al objetivo.

Este algoritmo se mueve siempre en la dirección cuya distancia sea la más cercana al objetivo. En el caso de haber algún obstáculo, el algoritmo buscará siempre a los nodos cuya distancia y dirección guie la búsqueda al objetivo, pero no siempre se trata de la más correcta, más cercana, en términos de distancia. Aun cuando la ruta más cercana pudiera ser la ruta b de la Figura 11; el algoritmo no tomaría el camino b, porque él siempre busca el camino más cercano al punto y tomando el camino b estaría yendo hacia atrás y se estaría alejando del objetivo, en vez de ir en dirección al objetivo.

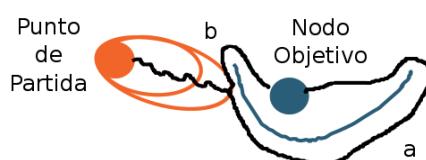


Figura 11. A modo de ilustración se muestra recorrido de un algoritmo con estrategia de búsqueda codicosa, mostrando dos posibles rutas, sin información adicional tomaría la ruta a, si solo se considera la distancia al objetivo.

Lo ideal sería diseñar un algoritmo que tomara las ventajas del algoritmo de búsqueda codicosa, el cual explora un número menor de nodos en muchos casos y el algoritmo de búsqueda de costo uniforme, el cual garantiza conseguir el camino de menor costo. El algoritmo A* permite tomar las ventajas de estos dos algoritmos ya descritos.

3.5.2. Búsqueda A*

El algoritmo A* (ver Figura 12) siempre expande los nodos cuyo valor de la función f es mínimo. La función f se define como:

$$f(\text{path}) = g(\text{path}) + h(\text{path})$$

donde,

$g(\text{path}) \rightarrow$ función de costo,

$h(\text{path}) \rightarrow h(\text{estado}) \rightarrow$ distancia del estado actual al objetivo

$f_1 = 75 + 374$. Costo de ir de Arad a Zerind más la distancia entre Zerind y Bucarest.

$f_2 = 140 + 253$. Costo de ir de Arad a Sibiu más la distancia entre Sibiu y Bucarest.

$f_3 = 188 + 329$. Costo de ir de Arad a Timisoara más la distancia entre Timisoara y Bucarest.

$$f_1=449; f_2=393; f_3=447$$

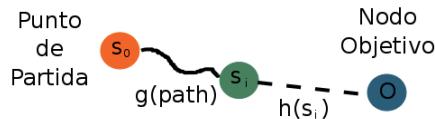


Figura 12. A modo de ilustración se muestra recorrido de un algoritmo con estrategia de búsqueda A*.

Minimizar la función g , garantiza conseguir el camino más corto, minimizar nos mantiene enfocados en el objetivo. El resultado de esta estrategia es la mejor posible, en el sentido que encuentra el camino más cercano, con el menor costo, expandiendo el mínimo número de nodos posibles con el menor costo. Veamos el ejemplo de aplicar esta estrategia al problema de conseguir la ruta más cercana y menos costosa desde Arad a Bucarest en el mapa de Rumanía.

En el problema tenemos definido el valor de las funciones de costo, lo cual indica la distancia entre las ciudades. Partiendo desde la ciudad de Arad. La función heurística será la distancia a vuelo de pajarito en línea recta entre cualquier ciudad y Bucarest en el mapa. Inicialmente:

$h(Arad) = 366$; $h(Zerind) = 374$; etc. representan en la Figura 13 los valores que están entre paréntesis. Por supuesto $h(Bucarest) = 0$, La distancia de Bucarest a Bucarest es cero.

Entonces, el primer valor a tomar sería aquel valor cuyo costo y distancia en línea recta al objetivo sea menor:

$f_1 = 75 + 374$; Costo de ir de Arad a Zerind más la distancia entre Zerind y Bucarest.

$f_2 = 140 + 253$; Costo de ir de Arad a Sibiu más la distancia entre Sibiu y Bucarest.

$f_3 = 188 + 329$; Costo de ir de Arad a Timisoara más la distancia entre Timisoara y Bucarest.

$$f_1=449; f_2=393; f_3=447$$

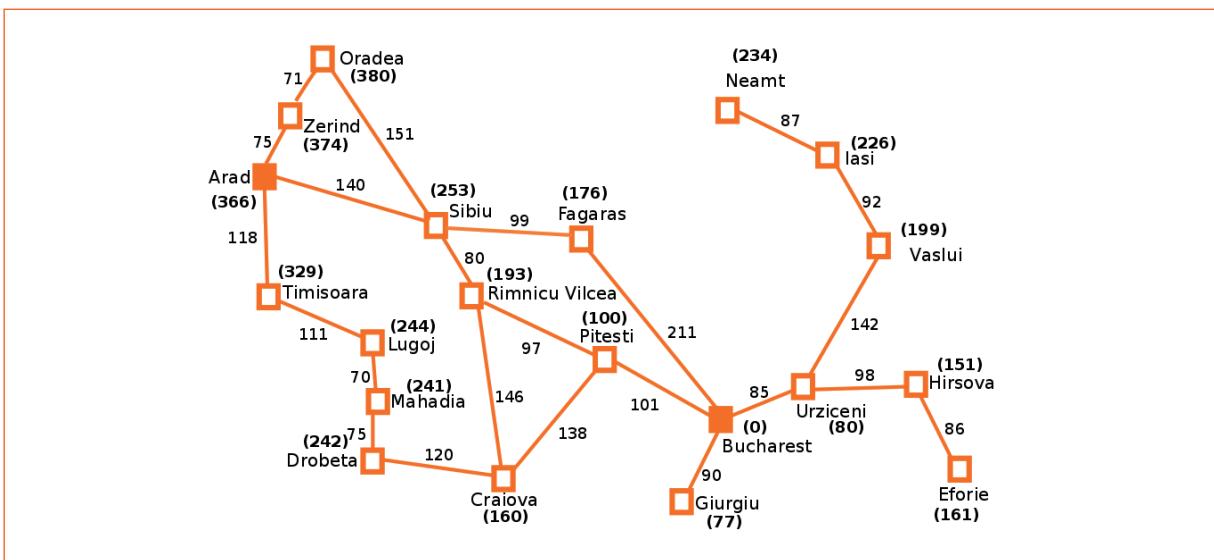


Figura 13. Mapa modificado de Rumanía, encontrando la ruta menos costosa desde Arad a Bucarest. Los valores entre paréntesis es el valor de la función heurística. Adaptado de Russell y Norvig (2010).

El algoritmo tomaría la ruta Arad-Sibiu como la más próxima a extender (f_2). Luego, desde el estado Sibiu, el más cercano en distancia de línea recta y menos costosa sería Rimnicu Valcea con:

$$f_1=671; f_2=415; f_3=413;$$

Donde f_1 sería el costo acumulado de ir de Arad a Oradea pasando por Sibiu más la distancia en línea recta de Oradea a Bucarest y f_2 el costo acumulado de ir desde Arad a Fagaras pasando por Sibia, más la distancia en línea recta entre Fagaras y Bucarest. Finalmente f_3 el costo de ir desde Arad a Rimnicu Valcea pasando por Sibia más la distancia en línea recta entre Rumnicu Valcea y Bucarest. Así el próximo estado a extender sería Rimnicu Valcea. Extendiendo el estado de Rimnicu Valcea, se observa que la ruta por Fagara tiene menor valor que las rutas por Rimnicu Valcea; Pitesti o Craiova. Así que se extiende entonces Fagara también y allí solo se tiene una ruta adicional a Bucarest y se alcanza Bucarest con una función f de 450 (140+99+211). Aun cuando ya se alcanzó el objetivo se necesita seguir verificando las próximas rutas, la que sería, en este caso, extendiendo a Pitesti. Allí existen dos caminos posibles, a Craiova y a Bucarest, resultando más cerca y menos costosa la ruta por Pitesti a Bucarest. Al comparar el costo con la otra ruta desde Arat por Fagaras, se obtiene que el camino de menor costo es la ruta Arad-Sibiu-Rumnica Valcea-Pitesti-Bucarest. El algoritmo A* ha logrado alcanzar entonces el camino con menor costo.

Se puede ver que el método de búsqueda heurística o A*, depende de la función heurística; si:

1. La función $h(s) <$ verdadero costo al objetivo desde el estado s , de tal manera que:

h no sobreestime la distancia al objetivo, además, h debe ser un valor optimista y admisible.

Cuando termina el algoritmo de A*, da como resultado una ruta cuyo costo estimado es c , el cual resulta ser el verdadero costo de la ruta, dado que $f = g + h$, y $h(\text{estado objetivo}) = 0$; entonces la función de costo quedaría $f = g$, donde g es el costo verdadero de la ruta encontrada.

Una función heurística optimista en el algoritmo A*, es la que permite alcanzar el camino más corto al objetivo. Ahora bien, este método funciona perfectamente en una estructura de árbol; para una estructura de grafos es ligeramente más complicado, sin embargo, intuitivamente funciona de forma similar.

3.6. Resolución de problemas de espacios de estados

Ya hemos evaluado el problema de espacios de estados con las ciudades de Rumanía, un problema bidimensional de estados físicos. Sin embargo, la tecnología para resolver problemas de búsqueda puede enfrentarse a diferentes tipos de problemas de espacios de estados. Un problema de espacios de estados es un problema que tiene múltiples configuraciones posibles. Problemas que pueden tomar en cuenta propiedades abstractas, no solo posición x,y en un plano. El problema de la aspiradora en un cuarto es un problema que tiene solo dos posiciones de estado (a y b) (ver Figura 14). Además, se observan solo dos posibles condiciones de estado de la aspiradora. La aspiradora puede estar *con sucio* o *sin sucio* dentro.

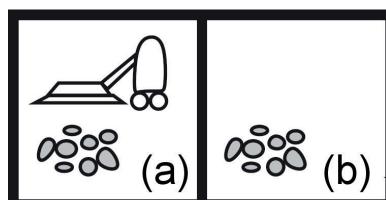


Figura 14. El problema de la aspiradora, hay dos cuartos, pueden estar limpios o sucios, la aspiradora pude moverse al cuarto de la derecha si está en el de la izquierda y viceversa. Puede aspirar y dejar limpio o puede ir al otro cuarto. Adaptado de Russell y Norvig (2010).

Es un problema que tiene dos estados posibles (a) o (b), cada uno de ellos puede estar sucio o no, para un total de 8 configuraciones posibles de estado. En la Figura se muestra el problema de la aspiradora con todos los estados posibles, donde los arcos denotan las acciones de cambiarse de un estado a otro, izquierda (I) o derecha (D) y la acción aspirar (A) (ver Figura 15).

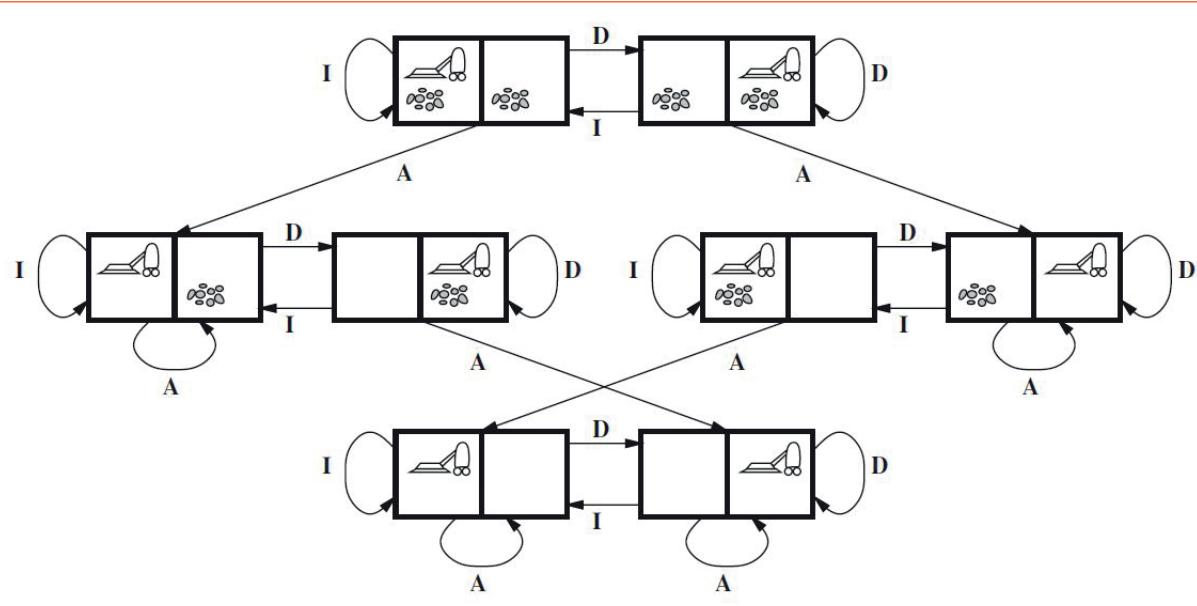


Figura 15. Configuraciones posibles del problema de la aspiradora en el cuarto sucio. Adaptado de Russell y Norvig (2010).

Supongamos que el estado inicial es el descrito en la Figura 14. Si movemos la aspiradora de la posición (a) a la posición (b), moviendo la aspiradora a la derecha (D) tenemos el nuevo estado:

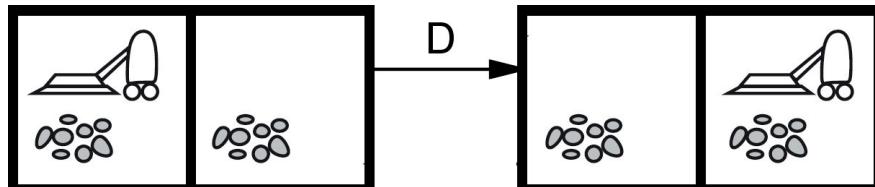


Figura 16. A partir del estado inicial se mueve a la derecha la aspiradora, generando un nuevo estado. Adaptado de Russell y Norvig (2010).

Si ahora a partir del nuevo estado se limpia el cuarto (A) se pasa entonces al nuevo estado:

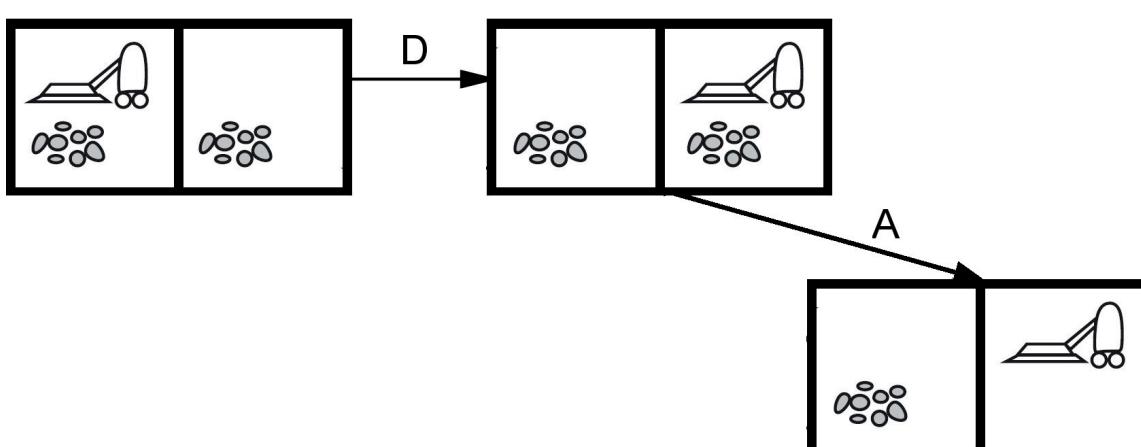


Figura 17. A partir del estado anterior de la Figura 16 se aspira el cuarto generando un nuevo estado. Adaptado de Russell y Norvig (2010).

Y así sucesivamente.

Si ahora hacemos el problema un poco más complicado y se le agregan algunas propiedades adicionales:

- a. A la aspiradora se le agrega un interruptor, el cual puede indicar tres posibles estados (*on/off/sleepy*).
- b. Además, la aspiradora tiene una cámara con sensor que verifica si hay sucio o no y puede estar encendida o apagada (dos estados posibles adicionales).
- c. El cepillo de la aspiradora tiene 5 posiciones posibles de altura (1/2/3/4/5).
- d. Para finalizar en vez de tener solo dos posiciones (a y b de la Figura 17) tiene 10 posiciones diferentes.

En este caso el número de estados posibles del problema sería:

$$2^{10} * 10 * 3 * 2 * 5 = 307.200$$

2^{10} , corresponde a las dos posibles condiciones; con sucio sin sucio, en las 10 posiciones, 10 posiciones posibles, 3 estados de encendido de la aspiradora, 2 estados del sensor, 5 posiciones del cepillo. Vemos como un problema trivial agregando algunas modificaciones en ciertas variables se puede convertir en un problema de 8 estados a un problema de miles de estados posibles, es por ello que se necesitan algoritmos eficientes de búsquedas en problemas de estados de procesos.

3.6.1. Problema del rompecabezas de bloques deslizantes

Otro de los problemas que puede ser resuelto con los algoritmos de búsqueda es el rompecabezas de bloques de 8 piezas. Se tiene un bloque de 3x3 espacios, en las que se tienen 8 piezas y un espacio de bloque disponible para mover las piezas adyacentes a dicho espacio (ver Figura 18). El objetivo es tener una cierta configuración con cierto orden y el estado inicial puede ser cualquier estado que no cumpla con el orden del estado objetivo.

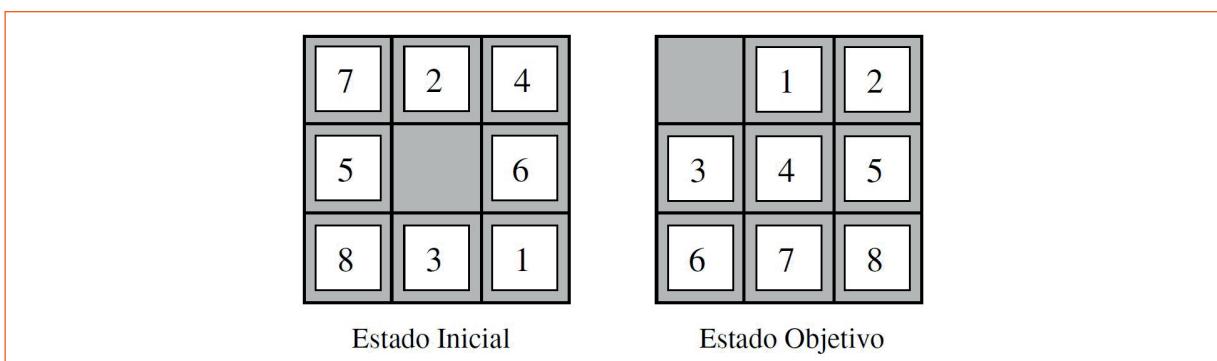


Figura 18. Problema del rompecabezas deslizante 3x3. Recuperado de Russell y Norvig (2010).

Asumamos una función heurística: $h_1 = \# \text{bloques colocados en posiciones incorrectas}$

Para el problema de la Figura 18, $h_1 = 8$. Todas las piezas están mal colocadas. Una segunda heurística podría ser $h_2 = \text{suma}(\text{distancias de bloques})$. Así que, si todos los bloques están en la posición correcta, $h_1 = 0$ y $h_2 = 0$. En cuyo caso dichas funciones heurísticas no estarían sobreestimadas. El problema consiste en determinar dichas funciones heurísticas, las cuales dependen de la naturaleza del problema. A medida que el número de piezas aumenta, en el caso del problema del rompecabezas deslizante, el algoritmo se vuelve más complejo (4x4, 5x5, etc.).

En este capítulo se describieron los algoritmos y métodos para resolver problemas de búsqueda en estructuras de árboles y grafos. Dichos métodos en general pueden encontrar el camino más corto y menos costoso al objetivo desde un estado inicial dado. Adicionalmente, dichos métodos pueden aplicarse de tal forma de que no se generen más caminos de los que se deben generar, evitando exploraciones innecesarias e inefficientes. Se pueden encontrar el número de caminos óptimos a generar, y esto es posible utilizando las funciones heurísticas que se pueden definir a partir del problema. Sin embargo, en este tipo de problemas de búsqueda de soluciones, todas las soluciones

que estos algoritmos pueden generar consisten en una secuencia fija de acciones, sin considerar obstáculos o diferentes situaciones que pudieran ocurrir en cada ruta definida. Todo esto funciona bien para los tipos de problemas vistos, pero solo si se cumplen ciertas condiciones. El dominio debe ser:

- a. **Completamente observable:** esto quiere decir que se debe conocer cuál es el estado inicial.
- b. **Conocido:** es decir se debe conocer el conjunto de opciones o configuraciones posibles.
- c. **Discreto:** debe tener un número finito de acciones a seleccionar.
- d. **Determinístico:** se debe conocer el resultado de tomar una acción.
- e. **Estático:** nada puede cambiar el mundo excepto por las acciones tomadas.

Si todas estas condiciones se dan, entonces se garantiza que el algoritmo va a funcionar y resolverá el problema. Más adelante veremos los casos en los que alguna de estas condiciones no se da.

4. Inferencia probabilística

La teoría de probabilidad nos explica como las redes bayesianas nos permiten representar una distribución conjunta completa. Una red bayesiana representativamente es un grafo dirigido acíclico, en el que cada nodo representa una variable aleatoria, con probabilidades distintas. Las redes bayesianas constituyen la base de los modelos de Markov (ya lo veremos en detalle más adelante). Las redes de Markov, a diferencia de las redes bayesianas, es un grafo no dirigido cíclico. Igualmente, las redes bayesianas incluyen la representación de independencia entre las variables aleatorias.

Se asume que el estudiante tiene claros los conceptos de probabilidad y redes bayesianas, tópicos fundamentales para entender este capítulo y que no son parte del alcance de este curso. Conociendo lo anterior explicamos en adelante cómo hacer inferencia probabilística. Veamos un ejemplo sencillo de una simple red bayesiana pero compleja, un ejemplo de evento inesperado, robo o terremoto (ver Figura 19).

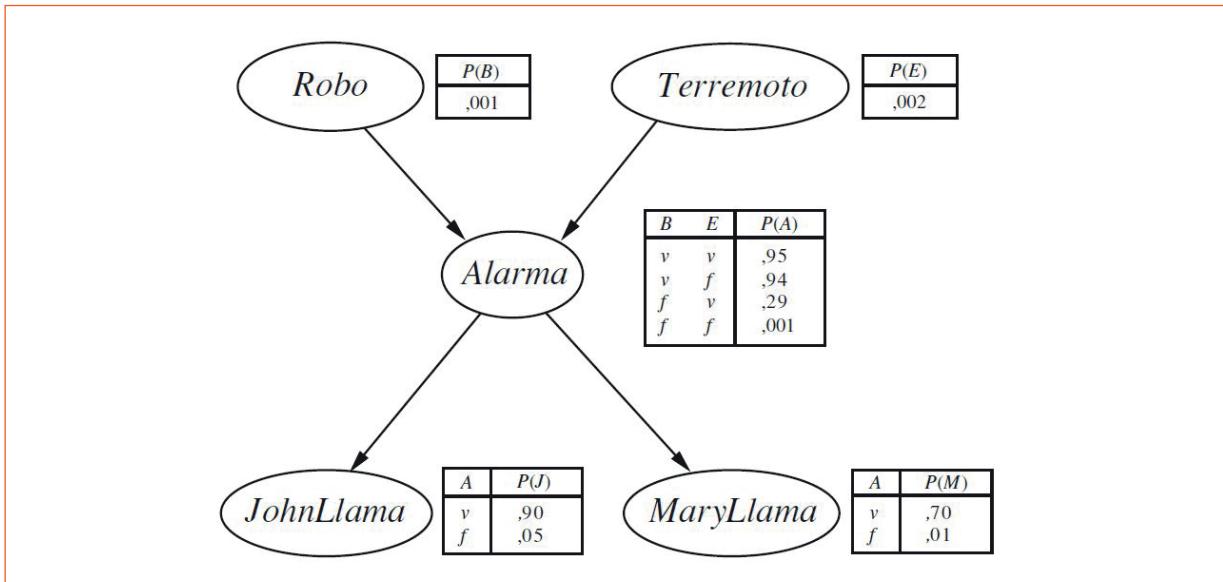


Figura 19. Ejemplo de red bayesiana típica pero compleja. Se muestra la topología de la red y las tablas de probabilidades condicionales Adaptado de Russell y Norvig (2010).

En el ejemplo Sara tiene una alarma antirrobo instalada en su casa, la cual es bastante sensible y confiable al detectar un robo, sin embargo, también se dispara cuando ocurre algún movimiento telúrico. La alarma se dispara cuando ocurre un robo (R) o un terremoto (T). Se tienen dos vecinos Juan (J) y Mary (M), quienes llaman a Sara al trabajo cuando escuchan la alarma. Juan siempre llama cuando escucha la alarma, sin embargo, a veces confunde el sonido de la alarma con el del teléfono o del timbre. Mary escucha todo el tiempo música muy alta y a veces no escucha la alarma, por lo tanto, no llama. En este ejemplo la alarma se dispara en caso de un robo o terremoto, pero Juan y Mary solo hacen la llamada si se dispara la alarma, es decir si se da el evento A ($a_i=\text{verdadero}$). Adicionalmente, debemos considerar que en este caso pueden existir otros factores como la música o el timbre de teléfono o de la casa, en el cual podrían afectar la acción de llamada de Juan o Mary, sin embargo, no son tomados en cuenta y pueden ser parte del conjunto de incertidumbres del sistema, en este caso no existe forma de evaluar dichos elementos, por lo que no son tomados en cuenta. Aun así, las redes bayesianas nos permiten estimar, con una probabilidad aproximada, la ocurrencia de algunos eventos dadas algunas evidencias.

¿Dadas las entradas R y T , cuáles serían los valores de J o M ? En inferencia estadística, generalmente se denomina a los nodos de entrada (R y T), variables evidencias y a los de salida (J y M), variables query. Las variables de las cuales se conocen los valores son evidencias y de las que queremos inferir un valor de salida le llamamos variables *query*. Todas las demás variables que no son ni evidencias ni *query* son variables escondidas. Dichas variables son aquellas de las cuales no se conocen los valores, y tampoco se necesita conocer para reportarlas, pero si se tendrán que calcular internamente. Un ejemplo simplificado de la red bayesiana se muestra en la Figura 20.

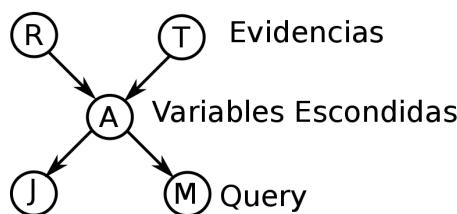


Figura 20. Ejemplo simplificado de la Figura 19.

En inferencia probabilística, la salida no es un valor singular para cada una de las variables *query*, sino más bien se refiere a una probabilidad conjunta. Así que la respuesta será una distribución probabilística para las variables *query*. La cual se denomina distribución posterior, dada las evidencias, y se pueden escribir como:

$$P(Q_1, Q_2, \dots, Q_i \mid E_1 = e_1, E_2 = e_2, \dots, E_j = e_j)$$

La probabilidad de una o más variables *query* dada la ocurrencia de algunas evidencias, que pueden ser desde cero evidencias hasta j evidencias. ¿De todos los valores posibles de todos los valores de las variables *query*, qué combinación de valores tiene la más alta probabilidad? Resumimos en la siguiente ecuación:

$$\text{argMax}_q P(Q_1 = q_1, Q_2 = q_2, \dots, Q_i = q_i \mid E_1 = e_1, E_2 = e_2, \dots, E_j = e_j)$$

Así podemos conocer qué valor de *Q* es máximo, dada las evidencias. Otra de las ventajas de usar redes bayesianas es que nos permiten no solo a partir de las evidencias (*R* y *T*) obtener los valores *query* (*J* y *M*), sino dados los valores *query* (*J* y *M*) como evidencia obtener los valores de entrada (*R* y *T*), incluso tener cualquier combinación posible de valores como evidencias y obtener el resto como valores *query* o de salida. Por ejemplo, se podría tener como evidencia los valores de *M* y los valores *J* y *R* como valores *query*. Imaginemos una situación en la que *M* reporta que *A* se activó y se quiere saber si ha sucedido a causa de *R* o no. En este caso, el nodo *M* sería el nodo evidencia y el nodo *R* un nodo *query*, mientras que los demás nodos serían nodos escondidos.

4.1. Inferencia por enumeración

Según las redes bayesianas cualquier probabilidad condicionada puede estimarse sumando los términos de la probabilidad conjunta completa.

Asumiendo que:

$$P(E = \text{verdadero}) = P(+e) = 1 - P(\neg e)$$

donde $P(\neg e) = P(E = \text{falso})$.

Inferencia por enumeración van por todas las posibilidades y se suman. Por ejemplo, ¿cuál es la probabilidad de que se haya disparado la alarma por un robo, dado que Juan y Mary hicieron la llamada?, esto es:

$$P(+b | +j, +m)$$

Sabemos que por definición de probabilidad condicional:

$$P(Q|E) = \frac{P(Q,E)}{P(E)}$$

entonces:

$$P(+b|+j,+m) = \frac{P(+b, +j, +m)}{P(+j, +m)}$$

Inferencia por enumeración toma la probabilidad condicional y se reescribe como probabilidad incondicional. Luego se enumeran todas las probabilidades atómicas y se calcula la suma de los productos. Así que $P(+b,+j,+m)$ puede ser determinada enumerando todos los posibles valores de las variables escondidas, en este caso hay dos a y e, quedando:

$$P(+b, +j, +m) = \sum_e \sum_a P(+b, +j, +m, e, a)$$

Ahora para obtener los valores atómicos, se debe reescribir la ecuación de tal forma que corresponda con las probabilidades condicionales asociadas a la red bayesiana.

$$\sum_e \sum_a P(+b, +j, +m, e, a) = \sum_e \sum_a P(+b) P(e) P(a|+b, e) P(+j|a) P(+m|a)$$

si:

$$P(+b) P(e) P(a|+b, e) P(+j|a) P(+m|a) = f(e,a)$$

entonces:

$$\sum_e \sum_a f(e, a) = f(+e, +a) + f(+e, \neg a) + f(\neg e, +a) + f(\neg e, \neg a)$$

Se transforma en la suma de una enumeración de términos, donde cada término es el producto de las probabilidades del conjunto de probabilidades completa de la red bayesiana.

Así pues:

$$f(+e, +a) = 0.001 * 0.002 * 0.95 * 0.9 * 0.7 = 0.000001197$$

$$f(+e, \neg a) = 0.001 * 0.002 * 0.95 * 0.05 * 0.01 = 0.95 \times 10^{-10}$$

$$f(\neg e, +a) = 0.001 * 0.998 * 0.94 * 0.9 * 0.7 = 0.59 \times 10^{-3}$$

$$f(\neg e, \neg a) = 0.001 * 0.998 * 0.06 * 0.05 * 0.01 = 0.29 \times 10^{-7}$$

Quedando. $P(+b, +j, +m) = 0.59 \times 10^{-3}$, el mismo calculo deberá hacerse con $P(+j, +m)$.

Una manera de acelerar el proceso del cálculo de inferencia por enumeración es aplicando el algoritmo de eliminación de variables.

4.1.1. Método de eliminación de variables

El algoritmo de eliminación de variables puede mejorar sustancialmente los cálculos necesarios para calcular las probabilidades condicionales en el cálculo de inferencias en redes bayesianas. Es un algoritmo de programación dinámica. Se realiza con una evaluación de la ecuación de izquierda a derecha (de arriba hacia abajo en la red bayesiana, ver Figura 21). Del ejemplo anterior tenemos:

$$P(+b|+j,+m) = \sum_e \sum_a P(+b) P(e) P(a|+b, e) P(+j|a) P(+m|a)$$

Extrayendo los valores independientes de la sumatorias internas quedaría de la siguiente forma:

$$P(+b|+j,+m) = a P(+b) \sum_e P(e) \sum_a P(a|+b, e) P(+j|a) P(+m|a)$$

Sin embargo, si tenemos una red más compleja con muchas más variables, escondidas, de salida o *query* y variables evidencias, obtenemos cálculos complejos y en muchos casos redundantes; la idea es minimizar la cantidad de cálculos. Para ello se pueden aplicar varios métodos, uno es maximizar la independencia de variables, a fin de poder extraer de los cálculos aquellas variables independientes y optimizar los cálculos, como se hizo en la ecuación antes descrita. El algoritmo de eliminación de variables funciona mejor que el de inferencia por enumeración en muchos de los casos. Requiere de conocimientos de álgebra para manipular los factores, usando arreglos multidimensionales, a fin de manipular los diferentes términos probabilísticos de la ecuación. Digamos por ejemplo que tenemos una red en la que se tienen tres nodos: el nodo *L*, una variable booleana que representa si llueve o no, la variable *T* indica si hay o no tráfico y la variable *D* el tiempo de demora en llegar a algún lugar, el cual depende si hay o no tráfico. Si llueve hay posibilidades de aumentar el tráfico y por ende se demorará más en llegar al destino (ver Figura 21).

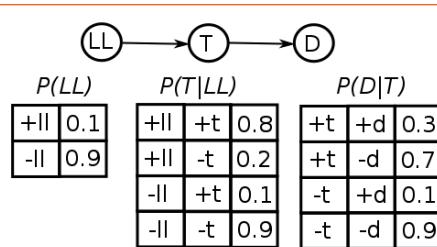


Figura 21. Ejemplo de la configuración de red bayesiana del problema del tráfico, lluvia y llega a destino.

La probabilidad de $P(+d) = \sum_{ll} \sum_t P(l) P(t|l) P(+d|t)$, se puede estimar usando el método por enumeración sin ninguna dificultad, sin embargo, en redes más complejas, el método de eliminación de variables nos permite calcular de manera óptima. El método de eliminación de variable permite combinar partes de la red en pequeñas partes, aplicar el método de enumeración de estas pequeñas

subredes, verlas como un nodo simple y continuar combinando. En este ejemplo se comienza por la red más grande y se eliminan algunas variables, realizando cálculos marginales sobre la subred y así tener ahora una red más pequeña. Este proceso se realiza en dos pasos, unión de factores y eliminación:

a. Unión de factores

Los factores son cada uno de los elementos de la matriz de probabilidades condicionales ($P(L)$, $P(T|L)$ y $P(D|T)$).

Para el ejemplo combinamos primero $P(L)$ y $P(T|L)$, la que sería la unión de las probabilidades de todas las variables en dichos factores. En este caso sería $P(L,T)$, para lo cual se crea la siguiente tabla (ver Tabla 1) como resultado de la unión.

Tabla 1

Tabla de probabilidades de la unión de probabilidades de los eventos lluvia (L) y tráfico (T)

$P(L, T)$		
$+l$	$+t$	0.08
$+l$	$-t$	0.02
$-l$	$+t$	0.09
$-l$	$-t$	0.81

La cual se construye a partir de las tablas $P(L)$ y $P(T|L)$, multiplicando los términos correspondientes cuando ocurran o no los eventos l y t. Por ejemplo, para la primera fila, $+l$ y $+t$, se multiplican los valores de $P(+l)*P(+l|+t)$, dando como resultado 0.08. Ahora la red se reduce a una red de dos factores: L y D (ver Figura 22).

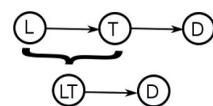


Figura 22. La red bayesiana del problema se transforma en una red más simple, combinando dos nodos en uno.

b. Eliminación

Ahora la matriz de probabilidades resultante de la unión se convierte en una matriz simple de probabilidades $P(T)$ (ver Tabla 2).

Tabla 2
Tabla de probabilidades resultante del proceso de eliminación.

$P(T)$	
$+t$	0.17
$-t$	0.83

Dicha la nueva matriz $P(T)$ se calcula sumando los valores en los que ocurre t , es decir $P(+t)=P(+l|+t)+P(-l|+t)$ y $P(-t)=P(+l|-t)+P(-l|-t)$. Ahora tenemos una nueva red generada a partir de la red inicial (ver Figura 23).

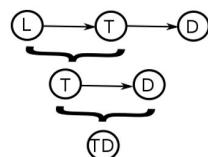


Figura 23. Se vuelve a unir los nodos restantes de la red de la Figura 22.

Se vuelve luego a unir T y D , estimando una nueva tabla $P(T,D)$ (ver Tabla 3).

Tabla 3
Tabla de probabilidades resultante del proceso de unión de las probabilidades $P(T)$ y $P(D)$.

$P(T,D)$		
$+t$	$+d$	0.051
$+t$	$-d$	0.119
$-t$	$+d$	0.083
$-t$	$-d$	0.743

De esta última tabla se puede calcular $P(D)$, $P(+d)$ and $P(-d)$, cuyo resultado entonces sería la sumatoria de las probabilidades de $P(T,D)$ cuando ocurre $d(+d \text{ o } -d)$, es decir $P(+d)=P(+t,+d)+P(-t,+d)$ lo mismo para $P(-d)$. Es así como funciona entonces el método de eliminación de variables, siendo un proceso continuo de unión de factores y eliminación de variables (ver Tabla 4). Si se hace una buena elección del momento de aplicar la unión de factores, el método de eliminación de variables puede llegar a ser más eficiente que el método simple por eliminación solamente.

Tabla 4
Tabla de probabilidades resultante del proceso de eliminación.

$P(L)$	
$+l$	0.134
$-l$	0.866

4.2. Inferencia aproximada

Digamos que queremos usar la unión de distribución de probabilidades, digamos la distribución de probabilidad de dos monedas, ambas monedas tienen probabilidad 0,5 de salir cara o cruz. Se puede construir una tabla en la que se vayan tomando muestras y contar cuántas caras o cruces por cada combinación de monedas van saliendo en cada intento. Se hace un número x de intentos, y podemos estimar la distribución de probabilidades observando las muestras. Para un número pequeño de muestras la estimación de la distribución de probabilidad puede no ser exacta. Puede que alguna variación aleatoria cause la no convergencia. En la medida que se tomen más muestras, la estimación de la distribución se aproximará a la real. Es así como el muestreo permite calcular el valor de inferencia aproximada sobre alguna variable.

En muchos casos puede no ser posible calcular inferencias como las inferencias exactas, donde se conoce la distribución de probabilidad de las variables, pueden ocurrir algunos casos en los que dicha distribución de probabilidad no se conoce. Para ello se puede calcular una probabilidad usando muestreos aleatorios, también conocidos como el **método de Monte Carlo**, el cual proporciona respuestas aproximadas a la distribución de probabilidad. La precisión de la estimación viene dada por el número de muestras, a mayor número de muestras el valor de probabilidad se aproxima al valor real de probabilidad. En este caso podemos mencionar dos familias de métodos:

- a. Métodos de muestreo directo
- b. Muestreo por las cadenas de Markov

4.2.1. Métodos de muestreo directo

El caso más sencillo de muestreo directo en redes bayesianas es muestrear en el orden topológico de la red la ocurrencia del evento. El valor de la muestra va a estar condicionado por los valores de distribución de probabilidad de sus padres. Veamos un ejemplo: se tienen cuatro eventos, tiempo nublado (N), se dispara el rociador de agua (R), puede que caiga lluvia (L) y la grama se moja (G), ver la Figura 24.

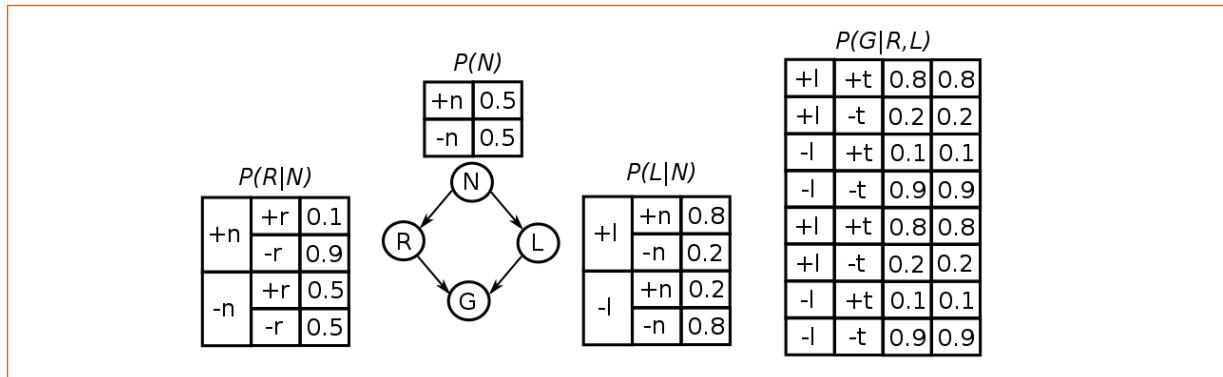


Figura 24. Muestra el ejemplo de red bayesiana con sus probabilidades, del ejemplo de la grama mojada por lluvia o por riego.

Supongamos el siguiente orden de ocurrencia: el día está nublado, el rociador no se dispara, llueve y la grama se moja, es decir una muestra: $+n, -r, +l, +g$. Podemos ver que se generan muestras a partir de la probabilidad *a priori*. Si el cielo está nublado $+n$, hay mayor probabilidad de que no se dispare el rociador de agua $-r$. Igualmente, si está nublado la probabilidad de que llueva es mayor. Dada la muestra $+n, -r, +l$, la probabilidad de que la grama se moje es $P(+g|-r, +l) = 0.9$ y $P(-g|-r, +l) = 0.1$. Esto indica que el modelo de muestreo es consistente, se puede usar esta clase de muestreo para calcular la distribución de probabilidad conjunta completa o para calcular el valor individual de las variables.

Muestreo por rechazo

Para calcular la probabilidad condicional, en algunos casos, se rechaza la muestra. Digamos que se quiere calcular la probabilidad condicional de $P(G|-n)$, la probabilidad de que se moje la grama dado que no está nublado. La muestra $+n, -r, +l, +g$, no nos sirve, dado que la probabilidad *a priori* es $+n$ y necesitamos estimar cuando no este nublado ($-n$). Entonces rechazamos la muestra $+n, -r, +l, +g$. Se ignora cualquier muestra que no corresponda con la probabilidad *a priori*, según la probabilidad condicional que se está calculando, manteniendo las que si corresponden. Digamos que la muestra obtenida es: $-n, +r, +l, -g$, la cual sigue siendo consistente.

Muestreo por ponderación de verosimilitud

El problema con el muestreo por rechazo es que, si la evidencia es poco probable, se terminan rechazando muchas muestras. Veamos el problema de la alarma. Donde tenemos variables para el robo y para la alarma y queremos calcular la probabilidad de que se haya dado un robo dado que la alarma se activó. Como es poco probable que se haya dado un robo, es muy probable que varias de las muestras sean del tipo $-r, +a$. No se dio un robo y la alarma se activó. Para evitar esto se utiliza el muestreo por ponderación de verosimilitud, que le asigna un peso a la probabilidad, en la que se fijan las variables evidencias. Es decir, en el ejemplo de la alarma, fijamos la evidencia *a priori* con el valor $+a$, la alarma se activó. Entonces se obtendría una lista de muestras como:

$$<-r, +a>, <-r, +a>, <+r, +a> \dots$$

Sin embargo, esto genera un problema, el resultado de las muestras es inconsistente, dado que se fija el evento $+a$. Para evitar la inconsistencia se le asigna una probabilidad a cada muestra y se ponderan

correctamente. En el muestreo por ponderación de verosimilitud vamos a recolectar muestras al igual que con el muestreo anterior, pero se le agrega un peso probabilístico a cada muestra. Digamos que en el ejemplo de la grama mojada se quiere calcular la probabilidad de lluvia dado que el rociador se disparó y la grama esta mojada $P(L|+r,+g)$. Digamos que en la muestra se tiene que el cielo está nublado $+n$. Se tiene la restricción para el cálculo de $P(L|+r,+g)$ que el rociador se activó. Entonces tenemos el valor probable cuando el rociador se activa y está nublado $P(+r|+n) = 0.1$. Este valor sería entonces el peso probabilístico de la muestra (ver Tabla 5).

Tabla 5

Construcción de probabilidades, dado que el cielo esta nublado y se activó el rociador, no sabemos de lluvia y grama.

Peso (w)	N	R	L	G
0.1	+	+	?	?

Veamos la variable Lluvia (**L**): en este caso no hay ninguna restriccion, los valores probables serían $P(+l|+n)=0.8$ o $P(-l|+n)=0.2$. Digamos que se muestrea con $+l$, tomando la de mayor probabilidad. Ahora evaluamos la variable **G**. Dado que el valor de sus ascendientes son todos positivos, puesto que estamos calculando $P(L|+r,+g)$ y según la tabla de las probabilidades condicionales $P(+g|+r,+l) = 0.99$, sería el nuevo peso ponderado a agregarle a la muestra y se multiplica por el peso probabilístico anterior, quedando el valor del peso probabilístico como $w=0.099$ (ver Tabla 6).

Tabla 6

Continuación de la construcción de las probabilidades de la tabla 5.

Peso (w)	N	R	L	G
0.099	+	+	+	+

Veamos la variable Lluvia (**L**): en este caso no hay ninguna restricción. Cuando se incluyen los pesos probabilísticos, y obtenemos una muestra como la del ejemplo, en la cual se obliga tener los valores $+r, +g$, con un peso probabilístico de 0.099, en vez de contarla como una muestra simple, se observa que los valores de muestreo por ponderación de verosimilitud son entonces consistentes. Sin embargo, no resuelve todos los problemas. Digamos que queremos calcular la probabilidad de que este nublado dado que se activó el rociador de agua, y llueve $P(N|+r,+l)$. En otras palabras, se restringen los eventos de lluvia y rociador a valores positivos. Dado que se usan las evidencias de los padres para estimar la probabilidad de ocurrencia de los nodos hijos, con esta técnica no habría manera de estimar la probabilidad del nodo padre, dado como evidencia la ocurrencia de los nodos hijos, así que no habría manera de calcular $P(N|+r,+l)$.

4.2.2. Muestreo por las cadenas de Markov (Muestreo Gibbs)

La técnica de muestreo de Gibbs fue llamada así en honor al físico Josiah Gibbs (Russell & Norvig, 2010). Dicha técnica es un caso especial del método de Monte Carlo para las cadenas de Markov, utilizado

para inferir en las redes bayesianas. Con este método se pretende estimar valores de distribución *a posteriori*. Para ello se crea una muestra inicial, se fija una de las variables y a partir de dicho valor de variable se estiman por probabilidades condicionales los valores de las otras variables. Cada nueva muestra dependerá de la muestra *a priori*. Generalmente este método es utilizado cuando no es tan sencillo generar muestras para simular valores de una distribución en redes bayesianas. A pesar de que las muestras sucesivas son dependientes de la anterior el método es consistente, dado que devuelve estimaciones consistentes de las probabilidades posteriores.

Digamos que del problema de la grama mojada se quiere resolver el siguiente interrogante: ¿Cuál es la probabilidad de que llueva, dado que el rociador se activó y la grama está húmeda? $P(L|+r,+g)$. Las variables evidencias R y G se fijan a los valores observados $+r$ y $+g$ y las otras variables N y L se inicializan de forma aleatoria. Asumamos que el resultado fue $+n$ y $-l$. El estado inicial sería:

$$<+n,+r,-l,+g>$$

Luego se fijan R y G y se muestran de N y L , a partir de los valores del manto de Markov (nodos padres, hijos y padres de los hijos). Se muestra N , dado los valores de su manto, sería $P(N|+r,+l)$. Lo mismo se hace con $P(L|N,+r,+g)$. Al muestrear N , supongamos que obtuvo el valor $-n$. Entonces al muestrear L , debemos evaluar $P(L|-n,+r,+g)$. Supongamos que se da el valor $+l$. La muestra siguiente sería entonces:

$$<-n,+r,+l,+g>$$

Y así, sucesivamente, sabiendo que en cada muestra se fijan valores a variables diferentes de manera aleatoria. Al final, cada muestra contribuye a resolver la interrogante inicial:

$$P(L|+r,+g)$$

Digamos que, como resultado de 80 muestras, 20 fueron $+l$ y 60 $-l$, quedando una relación $<20, 60>$, al normalizar las probabilidades serían $<0.25, 0.75>$.

Las demostraciones matemáticas de estos métodos no están al alcance de este manual, si quieren ahondar más en dichos métodos, pueden revisar el libro de texto recomendado de Russell y Norvig (2010).

5. Sistemas basados en reglas

Los sistemas basados en reglas o basados en conocimiento, son sistemas que utilizan como base la lógica clásica, aplicada sobre un conjunto de reglas. Las reglas son extraídas de una base de conocimiento, las mismas definen el problema. Se utiliza en este caso lo que se llama **motor de inferencia**, el cual, dadas las reglas, esa base de conocimientos, y aplicando lógica clásica permiten inferir y sacar conclusiones. Para la construcción de un sistema basado en reglas intervienen dos elementos muy importantes.

1. Los datos

Está conformado por el conjunto de evidencias o hechos conocidos. Es un elemento dinámico, cambia según el dominio del problema.

2. La base de conocimiento

Contiene el conjunto de objetos y reglas que permiten relacionar los objetos. Generalmente es estático, podría cambiar en los casos en que se incluyan elementos de aprendizaje.

5.1. Reglas

Las reglas incluyen dos partes, una premisa y una conclusión. Son del tipo Si 'X' entonces 'Y'. Cada regla es una afirmación lógica que relaciona dos o más elementos. Dichas reglas pueden estar conectadas por operadores lógicos. Principalmente las reglas están conformadas por dos partes:

1. Premisa

Es una expresión lógica entre las palabras claves Si y entonces. Se relacionan con otras premisas por medio de los operadores lógicos, y, o, o no.

2. Conclusión

Es la consecuencia de la ocurrencia positiva de la premisa. Si se cumple la premisa entonces ocurre tal o cual cosa.

5.2. Motor de inferencias

El motor de inferencia es quien procesa las reglas. Por lo tanto, usa los datos y el conocimiento para inferir sobre los datos, dadas las premisas y conclusiones y obtener nuevas conclusiones o hechos. Cuando se cuenta con el elemento de aprendizaje, es posible descubrir entonces nuevas premisas y conclusiones, que realimentaran la base de conocimiento. Está claro que si una premisa es cierta la conclusión también debe ser cierta (Si ocurre A entonces debe ocurrir B). En cuanto a motores de inferencias podemos distinguir dos tipos:

- **Modus ponens**

Es una máquina de inferencia simple, en la que se evalúa la premisa y si la misma es cierta entonces la conclusión pasa a ser parte de la base de conocimiento, pasa a ser un hecho, a ser verdad.

Si $\alpha \rightarrow \beta$, α . entonces β es cierto y pasa a ser parte de la base de conocimientos.

- **Modus tollens**

También es una máquina de inferencia sencilla, pero inversa a *Modus ponens*. En esta máquina de inferencia, si la conclusión no es cierta, entonces se asume que su premisa tampoco es cierta.

Si $\alpha \rightarrow \beta$, $\neg\beta$. Entonces, como la conclusión no es cierta la premisa tampoco es cierta, así que $\neg\alpha$ pasa a ser entonces parte de la base de conocimientos.

5.3. Mecanismos de resolución

Para la resolución de problemas usando las máquinas de inferencias, generalmente se siguen los siguientes pasos:

1. Se evalúan las premisas de todas las reglas según la base de conocimiento.
2. Si no es posible aplicar ninguna regla se termina sin éxito.
3. Si es posible aplicar alguna regla, se ejecuta y espera el resultado.
4. Si se llega al objetivo se termina con éxito, si no, se siguen aplicando las reglas.

Para eso se examinan los hechos y las reglas. Es posible añadirle nuevas reglas o a partir de las reglas de la base de conocimiento generar nuevas reglas. **El motor de inferencia debe decidir el orden de aplicación de las reglas.** Se aplican entonces las reglas lógicas *modus ponens* y *modus tollens*. Se da lo que conocemos como **encadenamiento de reglas**.

5.4. Encadenamiento de reglas

Es una estrategia donde se aplican las reglas lógicas *modus ponens* y *modus tollens*, para la aplicación de reglas en motores de inferencias. Esta estrategia de encadenamiento de reglas se aplica cuando las premisas y conclusiones de algunas reglas coinciden con las conclusiones de otras. Esto permite, de hecho, crear nuevos hechos. El encadenamiento se repite hasta no poder obtener más conclusiones dados los hechos iniciales y los hechos creados a partir de la aplicación de las reglas lógicas. Veamos el siguiente ejemplo:

- Regla 1. Si **A** y **B**, entonces **C**
- Regla 2. Si **D**, **E** y **F**, entonces **G**
- Regla 3. Si **H** e **I**, entonces **J**
- Regla 4. Si **C** y **G**, entonces **K**
- Regla 5. Si **G** y **J**, entonces **L**
- Regla 6. Si **K** y **L**, entonces **M**

Asumiendo que tenemos como hechos: **H**, **I**, **K** y $\neg M$, usando las dos reglas lógicas de inferencia, *modus ponens* y *modus tollens*, tenemos:

- a. Por la regla 3, tenemos que **J** es cierto, aplicando *modus ponens*.
- b. Por la regla 6, como tenemos que $\neg M$, usando *modus tollens* $\neg(K \text{ y } L)$, pero **K** es cierto, así que entonces $\neg L$, es decir que **L** debe ser falso.
- c. Por la regla 5, **J** es cierto, **L** es falso, entonces **G** debe ser falso, es decir $\neg G$. Aplicando nuevamente la regla *modus tollens*.

5.5. Encadenamiento de reglas basadas en objetivo

En este tipo de encadenamiento se fija previamente cuál es la conclusión objetivo a la que se quiere llegar. Las reglas lógicas son aplicadas orientadas a la obtención de dicha conclusión. Si no es posible obtener la conclusión objetivo entonces el sistema requerirá de información adicional por parte del usuario. Veamos el ejemplo:

Tenemos como hechos que: $\{D, E, F, L\}$ son ciertos. Y tenemos las siguientes reglas:

- Regla 1. Si A y B , entonces C
- Regla 2. Si D, E y F , entonces G
- Regla 3. Si H e I , entonces J
- Regla 4. Si C y G , entonces K
- Regla 5. Si G y J , entonces L
- Regla 6. Si K y L , entonces M

Se fija como nodo objetivo el nodo M .

Entonces se fijan como objetos marcados los hechos más el nodo objetivo $\{D, E, F, L, M\}$.

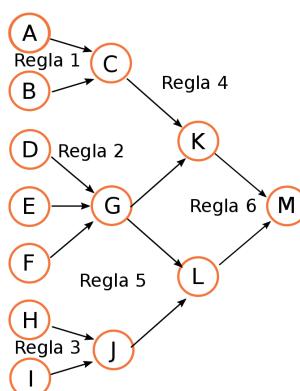


Figura 25. Estructura de árbol o red bayesiana construida a partir de las reglas del ejemplo en estudio.

Se buscan las reglas que incluyan los objetivos, sería la regla 6. Sin embargo, no se puede concluir porque no se conoce el valor de K , el cual debe ser cierto, entonces se marca K como objetivo previo al objetivo M . Quedando el objetivo: $\{D, E, F, L, M, K\}$.

Evaluamos ahora la regla 4 que contiene el objetivo K . Dicha conclusión aún no se puede dar si no conocemos los valores de C y G . Se agregan entonces como objetivo. El objetivo G podemos obtenerlo a partir de la regla 2. Aplicamos la regla 2 y obtenemos que G es cierta. De C no podemos decir nada aún. Se agrega entonces al objetivo: $\{D, E, F, L, M, K, C\}$.

Para que se cumpla **C** se deben cumplir **A** y **B**, pero no se sabe nada de estos. Por lo tanto, uno de ellos se puede sumar al objetivo: $\{D, E, F, L, M, K, C, A\}$.

Se busca, entonces, alguna regla que incluya a **A** en su conclusión y no se tiene ninguna, entonces se le debe preguntar al usuario sobre el valor de **A**. Asumiendo que **A** es cierta, se va hacia atrás, se vuelve a tocar el nodo objetivo **C**. Se tiene que **A** es cierto, pero no se sabe nada de **B**. Se incluye **B** en los objetivos: $\{D, E, F, L, M, K, C, A, B\}$.

Se verifica alguna regla cuya conclusión es **B**: no hay ninguna. Entonces se pregunta al usuario por **B**; asumamos que es cierta. Entonces se concluye por la regla 1 que **C** es cierta. Vamos atrás y tenemos a **K** como nodo objetivo. Por la regla 2 **G** es cierta, y sabemos ya que **C** es cierta entonces por la regla 4, **K** es cierta. Volvemos atrás y volvemos a tomar a **M** como objetivo. Sabemos que **K** es cierta y que **L** también es cierta entonces por la regla 6, **M** es cierta.

- **Encadenamiento hacia adelante**

Se busca solucionar el problema a partir de las reglas cuyas premisas contengan los objetivos.

- **Encadenamiento hacia atrás**

Se busca solucionar el problema a partir de las reglas cuyas conclusiones contengan los objetivos.

6. Planificación

Hemos definido a la IA como el área de estudio de procesos para encontrar las acciones apropiadas de un agente inteligente. Es así como podemos ver que **la planificación puede ser el núcleo de la IA**. Hemos visto técnicas de búsqueda sobre un espacio de estados usando técnicas como la técnica de A*. Dado el espacio de estado y la descripción del problema, podemos encontrar una solución, encontrando el camino óptimo al objetivo. Dichas técnicas funcionan bastante bien para una gran variedad de ambientes, sin embargo, solo funciona cuando el ambiente es determinístico y completamente observable.

6.1. Resolución de problemas vs planificación

En la resolución del problema de búsquedas, partimos desde un punto de inicio y llegamos a un punto objetivo, teniendo un espacio de estados determinístico y completamente observable. Aplicando alguno de los métodos de resolución de problemas de búsqueda, encontramos un camino y es óptimo.

Veamos el ejercicio en el que tenemos el mapa de Rumanía y encontramos el camino más cercano a Bucarest. Como turistas sin conocer las vías, tomamos la que nos parece más cercana, para llegar en menos tiempo. Al estudiar el mapa y encontrar el camino más cercano, comenzamos a ejecutar la solución y seguir la ruta que en principio seleccionamos como la óptima. Sin embargo, si no conocemos el camino, pueden pasar dos cosas: que el camino sea agradable o que no lo sea. Esto

no habrá manera de saberlo si no vamos de la mano con la planificación y la ejecución. Es decir que deberíamos tener cierta interacción con el ambiente para ir ejecutando en la medida que vamos planificando.

6.2. Planificación vs ejecución

Algunos ambientes pueden tener propiedades que hacen difícil manejar correctamente la planificación sin improvisto y luego hacer la ejecución. Pero no siempre es así. Los ambientes pueden tener propiedades que dificultan encontrar la solución. Algunas de esas propiedades son:

1. **Ambientes estocásticos:** cuando no sabemos cuál será el resultado de tomar una acción. En cuyo caso se debe tener algún plan de contingencia por si lo planeado no sale como queremos.
2. **Ambientes multiagentes:** en cuyo caso no se conoce cuál puede ser el resultado de la acción del otro agente. Se debe planear cuáles son las acciones que podría tomar el otro agente, y debemos reaccionar cuando los otros agentes hacen algo que no se espera. Esto solo podemos saberlo en el tiempo de ejecución y no en el tiempo de planificación.
3. **Ambientes parcialmente observables:** digamos que se tiene una ruta planificada de ir del estado **A**, pasando por **S**, **F**, y llegar a **B**. Resulta que el camino de **S** a **F** está cerrado y no hay manera de saberlo si no hasta llegar al punto **S**. Al principio no se tiene dicha información.
4. **Desconocimiento:** adicionalmente a las dificultades anteriores que tienen que ver con el ambiente, es posible que tengamos dificultades en la falta de conocimiento, es decir parte del mundo del problema es desconocido. Por ejemplo, tenemos un mapa o un GPS, pero no son seguros o están incompletos. No seremos capaces de ejecutar la ruta que se había planificado, sino hasta el momento de comenzar a ejecutarlo.
5. **Planes Jerárquicos:** es posible, en algunos casos, que se deba trabajar con un plan jerárquico. En el caso de ir del punto **A** al **B**, pasando por **S** y **F**. La planificación se hizo en un nivel alto, solo indicando los puntos que guían la ruta, sin embargo, a bajo nivel tenemos el carro, aceleramos, frenamos, son también acciones, pero de bajo nivel. En cuyo caso en vez de planificar sobre el espacio del mundo de estados, se planifica sobre el espacio de estados de creencias.

Usando el ejemplo del problema de la aspiradora (ver Figura 15):

En cada estado tenemos dos lados, el **A** el lado izquierdo y el **B** el lado derecho. En dicho ambiente hay una aspiradora, en ambos cuartos puede que esté limpio o sucio. Si está limpio no hace nada y se pasa al otro lado. Si está sucio limpia etc. Tiene un total de ocho estados posibles. En la Figura 15 están todos los estados posibles. Supongamos ahora que al robot aspiradora se le dañaron los sensores; ahora no puede saber en qué lado está del cuarto ni si está sucio o limpio. Ahora se tiene un mundo inobservable o un mundo con pérdida de sensores. Una manera de planificarlo es conocer todos los estados posibles, en este caso son 8 (ver Figura 26 (a)). Aun sin sensores, asumiendo que la aspiradora está en la izquierda y se mueve a la derecha, tendríamos solo los estados en que la aspiradora pasa al lado derecho

(ver Figura 26 (c)), con cuatro estados. Lo mismo si asumimos que está en el lado derecho y nos movemos a la izquierda (ver Figura 26 (b)). Lo mismo sucede si estamos en el estado derecho (ver Figura 26 (c)) y de ese estado pasamos al lado izquierdo, viceversa también nos encontramos con dichos estados. Observemos que de igual manera se puede crear un plan que alcance el objetivo sin aun observar el mundo completo, un mundo parcialmente observable, para ello debemos conocer todas las posibilidades de posibles estados. Este tipo de plan es llamado **Plan conformista o planificación sin sensores**.

Ahora bien, si el objetivo es estar en una localidad limpia, tendríamos los casos mostrados en la Figura 26 (d). Los estados a partir de un estado conocido, es decir se sabe el estado actual pero no mucho del ambiente, se crean estados posibles a partir del estado actual, lo cual reduce el número de estados posibles para seguir adelante. Estos estados son llamados **estados de creencias**. Es así como del estado de la Figura 26 (a), podemos pasar al (b), (c) o (d), aun sin tener conocimiento de la información que pudieran dar los sensores, que en este ejemplo se asume que no existen sensores.

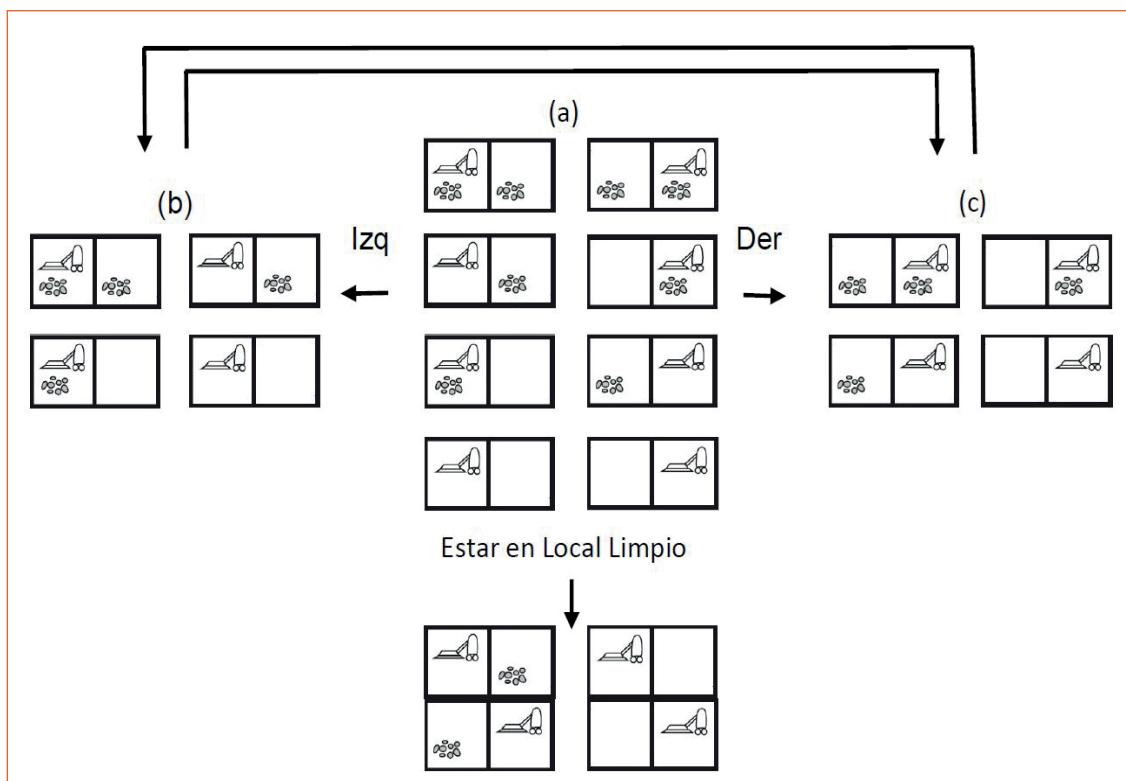


Figura 26. Estados posibles o creencia, en función de otros posibles estados conocidos o creencias.

A partir de estos estados podemos seguir a otros estados, siguiendo las acciones, moverse a la izquierda, derecha o estar en localidad limpia. Este ejemplo es un ejemplo de un ambiente sin sensores y determinístico. En un ambiente estocástico cualquier plan que se diseñe puede ser que ayude a llegar al objetivo.

6.3. Planificación en ambientes no deterministas

En el mundo real se pueden definir cuatro tipos de planificación para trabajar en ambientes indeterminados:

1. **Planificación sin sensores o conformistas:** son planes secuenciales diseñados para ser ejecutados sin percepción. Este tipo de planificación asegura que se puede lograr alcanzar el objetivo en todos los posibles estados. Frecuentemente este tipo de planificación es inaplicable.
2. **Planificación condicional o de contingencia:** se construye un plan condicional con diferentes ramas para llevar a cabo las diferentes contingencias que puedan surgir. El agente planifica y ejecuta, evaluando las condiciones apropiadas. Este tipo de planificación trabaja con indeterminación ilimitada, es decir, las precondiciones posibles o efectos o son desconocidos o son demasiados extensos como para enumerarlos.
3. **Vigilancia de ejecución y replanificación:** con este enfoque el agente puede usar cualquiera de las técnicas de planificación presentadas en los puntos 1 y 2 para construir un plan, vigilando siempre el resultado de las acciones tomadas, verificando si me permite llegar al objetivo o si requiere de una revisión. En cuyo caso requiere de una replanificación.
4. **Planificación continua:** las técnicas de planificación generalmente paran cuando el objetivo es alcanzado. En los agentes con planificación continua, están en constante funcionamiento. Al igual que los otros planificadores pueden encontrarse con eventos inesperados, incluso puede abandonar el objetivo y agregar objetivos adicionales.

6.4. Planificación clásica vs planificación no-clásica

La planificación clásica es aquella que es realizable en entornos completamente observables, deterministas, finitos, estáticos y discretos. En la **planificación no-clásica, se utilizan en entornos parcialmente observables o estocásticos, son los llamados planificación bajo incertidumbre.** Ya vimos varios algoritmos de búsqueda, donde los resultados consisten en tener una secuencia ordenada de acciones. En el proceso de planificación no necesariamente es así. En la planificación clásica o no clásica se aprovecha de la información proporcionada por la estructura del problema para encontrar el mejor plan para conseguir el objetivo.

Para la planificación se deben representar:

1. **Los estados:** son representados por hechos, o conjunciones de hechos.
2. **Los objetivos:** igualmente es una representación de hechos o conjunciones de hechos, donde hay un ejemplo s que satisface el elemento objetivo. Este puede tener uno o más hechos necesarios para el cumplimiento del objetivo con otros hechos adicionales.
3. **Las acciones:** es representada por un conjunto de precondiciones que deben cumplirse antes de ser ejecutada y las consecuencias de ejecutarla.

6.5. Encontrar la planificación exitosa

El proceso de encontrar un plan que funcione puede ser hecho a través de la búsqueda, como vimos en el apartado sobre resolución de problemas. Recordemos que, en resolución de problemas, se inicia en un estado, un estado simple, no un estado de creencia. Luego se hace una búsqueda en un árbol de búsqueda y se construye una estructura de árbol (estructura triangular) con todos los posibles estados que hemos estado buscando. Luego se encuentra una ruta que permite llegar al estado objetivo. Encontramos entonces una ruta simple dentro de la estructura de árbol. Con estados de creencias durante la planificación también se tiene una estructura de árbol. Un plan es exitoso en la medida en que con su ejecución se logre obtener o encontrar el objetivo.

En problemas de soluciones ilimitadas la garantía de que se pueda encontrar un plan exitoso es que cada hoja del árbol pueda ser un objetivo. En problemas de soluciones acotadas, es posible tener múltiples ramas del árbol, pero las ramas cíclicas deben evitarse a fin de garantizar un plan exitoso, evitando los ciclos en la búsqueda en la estructura de árbol.

6.6. Planificación con búsqueda de espacios de estados

Básicamente los algoritmos que se discutieron con anterioridad son exactamente los mismos que se usan para la planificación en la búsqueda de espacios de estados. La diferencia es que, en el espacio de estados puede haber en un mismo estado o nodo diferentes configuraciones de estados. Entonces los métodos que se definieron en esa sección de resolución de problemas de búsqueda son exactamente los mismos que se utilizan en la planificación clásica.

6.6.1. Búsqueda hacia adelante (Progresión)

La técnica de planificación con búsqueda de espacios de estados hacia adelante es similar a las que se aplicaron en la sección de resolución de problemas de búsqueda. Se comienza en un estado inicial, luego le sigue un conjunto de acciones que le permiten llegar al objetivo. Entonces se tienen:

1. Estado inicial
2. Objetivo
3. Conjunto de Acciones
4. Función de costo

6.6.2. Búsqueda hacia atrás (Regresión)

Este tipo de búsqueda también fue descrita con anterioridad en el apartado sobre resolución de problemas. Lo más complejo de la búsqueda hacia atrás es que no siempre es obvio generar una descripción de los posibles estados que preceden al estado o nodo objetivo. En este caso vamos desde el objetivo identificando los posibles estados que preceden al mismo. La ventaja es que permite considerar solo acciones relevantes que me llevarán a cumplir el objetivo, considerando que una acción es relevante para una secuencia de objetivos si alcanza un conjunto de ellos. El plan

por regresión parte desde el objetivo, entonces vamos buscando cuales son los posibles estados que pueden ser predecesores que resultaran en el estado actual. Si partimos desde el objetivo, el estado actual sería tratado como objetivo parcial del problema de búsqueda hacia atrás.

7. Planificación bajo incertidumbre

El proceso de planificación bajo incertidumbre es una de las áreas quizás más interesantes y complejas (ver Figura 27). En robótica es muy importante planificar bajo incertidumbre, planificar las acciones de un agente en un mundo real lleno de incertidumbres, algunas manejables y otras no tanto. En ambientes bajo incertidumbre se cuenta entonces con percepciones. Es importante percibir mediante sensores o algún otro elemento lo que está sucediendo en el entorno o el estado en que se encuentra el agente. La información que llega a percibir el agente puede ser incompleta o incorrecta, lo cual dificulta tener una clara descripción del estado actual. Es posible que, al percibir cierta información, el agente deba modificar o reemplazar el plan en caso de que algo inesperado suceda.

Es importante saber también que el grado de conocimiento correcto depende de cuánta indeterminación hay en el mundo real. En algunos casos el grado de indeterminación puede llevar al agente a tomar acciones impredecibles. La robótica está llena de incertidumbre. Esta sección de planificación bajo incertidumbre es muy importante para el desarrollo de agentes inteligentes en la robótica.

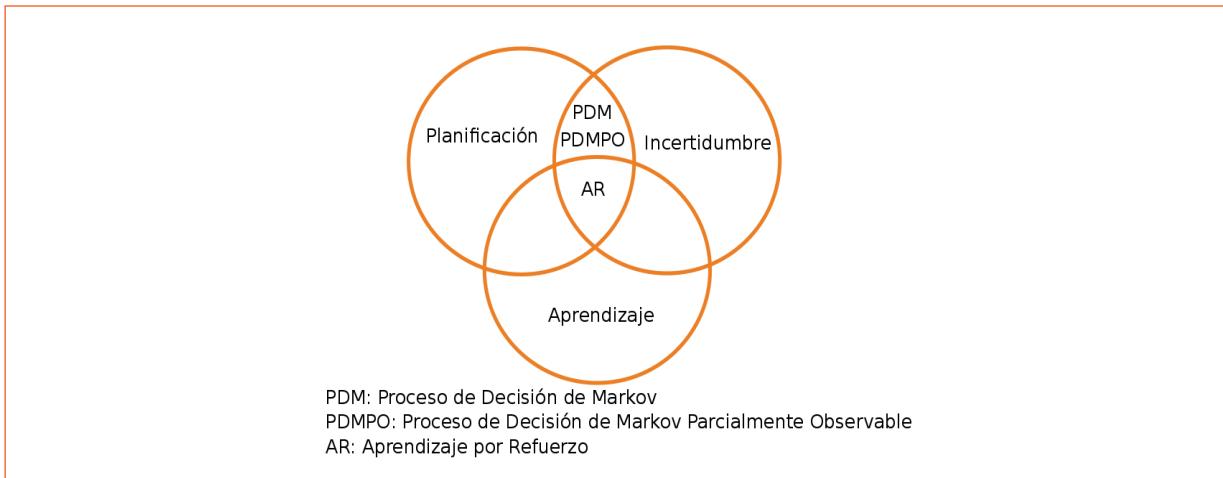


Figura 27. Relación entre planificación, incertidumbre y aprendizaje.

En la Tabla 6 se muestran los algoritmos que se aplican en diferentes entornos, completamente observables vs parcialmente observables, determinísticos vs estocásticos.

Tabla 6
Algoritmos vs entornos donde se aplican

Peso (<i>w</i>)	<i>N</i>	<i>R</i>	<i>L</i>	<i>G</i>
0.099	+	+	+	+

Recordemos que, en ambientes determinísticos, se conoce *a priori* el resultado de cada acción a ejecutar y el entorno es conocido. Mientras que, en un entorno estocástico, variables aleatorias pueden influir en el estado del mismo en cualquier momento, así mismo el resultado de las acciones puede llevar a resultados o cambios de estados inesperados, o una variante de estados. En ambientes completamente observables, se conoce *a priori* el ambiente y sus diferentes aristas, mientras que, en un entorno parcialmente observable, solo podemos ver limitadamente cuando se llega al estado actual y aún si se percibe el entorno.

Por lo general se definen cuatro métodos para trabajar con indeterminación. En casos de indeterminación limitada e indeterminación ilimitada.

7.1. Planificación sin sensores

De este tipo de planificación ya habíamos hablado con anterioridad, también es llamada **planificación conformista**. Este tipo de planificación se basa en la evaluación de estados creencias, la cual consiste en tener todas las posibles configuraciones de estados posibles en cada nodo. Todo plan debe asegurar que alcanza el objetivo de todas las posibles circunstancias. En la planificación sin sensores no se tiene sino **información parcial del entorno del estado actual**. Normalmente trabaja bajo la coerción, con el objetivo de condicionar el comportamiento del agente. Pero no siempre es posible, por lo que este tipo de planificación mayormente no es aplicable.

7.2. Planificación condicional

La planificación condicional es también llamada **planificación contingente**. Inicialmente se realiza un plan y el agente comienza a aplicarlo y va continuamente evaluando las distintas posibilidades, creando ramas. Inicialmente se crea un plan con diferentes ramificaciones de posibilidades de acciones, y comienza a observar el plan e ir ejecutando y evaluando.

7.3. Planificación con vigilancia y replanificación

En este enfoque el agente utiliza cualquiera de los enfoques de planificación anteriores y construye un plan. **Constantemente está en proceso de validaciones o verificación del resultado de las acciones a fin de verificar si el plan tiene la capacidad de cumplir con el objetivo dado el estado actual en el que se encuentra.** Mientras todo vaya bien y desde el estado actual se tenga la capacidad de acercarse al objetivo, no pasa nada. La replanificación ocurre cuando en uno de los estados actuales se llega a un punto donde se puede percibir que desde allí no se llega al objetivo. De esta manera puede encontrar otra forma de llegar al objetivo.

7.4. Planificación continua

Un planificador continuo a diferencia de los planificadores vistos está en continua planificación, y no se detiene cuando se consigue el objetivo. **Está en constante formulación de nuevos objetivos u objetivos adicionales, incluso puede abandonar los objetivos iniciales.** Esto suele ocurrir si se presentan circunstancias inesperadas del entorno.

7.5. Planificación multiagente

La planificación multiagente requiere de elementos de coordinación, y colaboración para trabajar en función de cumplir un mismo objetivo u objetivo común. Debe haber una cooperación en la ejecución de planes y objetivos conjuntos. Para ello los agentes deben tener conocimiento en común, manejar la misma información a fin de coordinarse.

Algunos mecanismos de coordinación son:

- **Convención:** se trata de convenir en algunas acciones o decisiones en común. Por ejemplo: "Un jugador siempre se debe encontrar del lado de la red", en un juego de tenis, por ejemplo.
- **Leyes sociales:** algunas convenciones se vuelven más bien leyes sociales, Por ejemplo: "Conducir del lado correcto de la carretera". Las convenciones siempre dependerán del dominio y las restricciones.
- **Agente pájaro:** cuando las convenciones simulan el comportamiento de los pájaros por ejemplo coinciden en tres reglas:
 - **Separación:** alejarse de los vecinos cuando comienzan a acercarse.
 - **Cohesión:** moverse en posición promedio cerca del grupo.
 - **Alineamiento:** moverse en sentido y orientación del grupo.

- **Comportamiento emergente:** cuando todos los agentes ejecutan la misma política.
- **Reconocimiento de Planes:** cuando una acción determina un plan conjunto sin ambigüedades.
- **Intención Conjunta:** para que haya una buena coordinación entre los agentes es necesario e indispensable que haya una intención continua entre los agentes que ejecuten el plan conjunto.
- **Conflicto:** los agentes multiagentes no siempre deben estar coordinados para la ejecución conjunta del mismo plan, en algunos casos si son multiagentes oponentes, entran en conflicto.

7.6. Procesos de decisión de Markov

Los procesos de decisión de Markov permiten realizar la planificación bajo ambientes de incertidumbre. La manera más sencilla de especificar un proceso de decisión de Markov es a través de un grafo. Supongamos que se tiene el siguiente grafo (ver Figura 28):

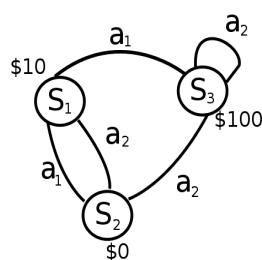


Figura 28. Grafo para mostrar el conjunto de estados, acciones, y recompensa por cada estado.

Un proceso de decisión de Markov, tiene un conjunto de estados $S=\{S_1, S_2, \dots, S_n\}$, un conjunto de acciones $a=\{a, a_2, \dots, a\}$, la matriz de transición de estados $T(S, a, S') = P(S'|aS)$ y el objetivo podría ser encontrar la ruta con la mayor recompensa $R(S)$. Cada estado tiene una recompensa de estar en cada uno, en nuestro caso el estado S_1 es de $\$10$, el de S_2 es $\$0$ y el de S_3 es $\$100$. La idea es hacer una búsqueda que dé como resultado el máximo rendimiento.

8. Aprendizaje automático (*Machine Learning*)

El aprendizaje automático o aprendizaje de máquina es un área de la inteligencia artificial que estudia la manera como los procesos aprenden a resolver problemas en función de la experiencia obtenida en base a los datos y observaciones en el comportamiento de los datos en el pasado (ver Figura 29). Hoy en día el mundo se está enriqueciendo con información inmensurable, que se encuentra digitalmente almacenada y que puede ser procesada de manera abierta o cerrada. El aprendizaje automático es un conjunto de técnicas de aprendizaje a partir de la observación de datos, ejemplos o de entrenamiento. Métodos que nos permiten inferir o predecir comportamientos *a posteriori*, según las observaciones previas dadas. El objetivo principal del aprendizaje automático es estimar una función que permite modelar el comportamiento de los datos, de tal manera que al ingresar un conjunto de datos nuevo pueda determinar o predecir el resultado más acertado. En otras palabras y según Samuel Arthur (1959) es la rama de la inteligencia artificial cuyo objetivo es desarrollar técnicas que le permitan al computador aprender sin ser programado explícitamente.

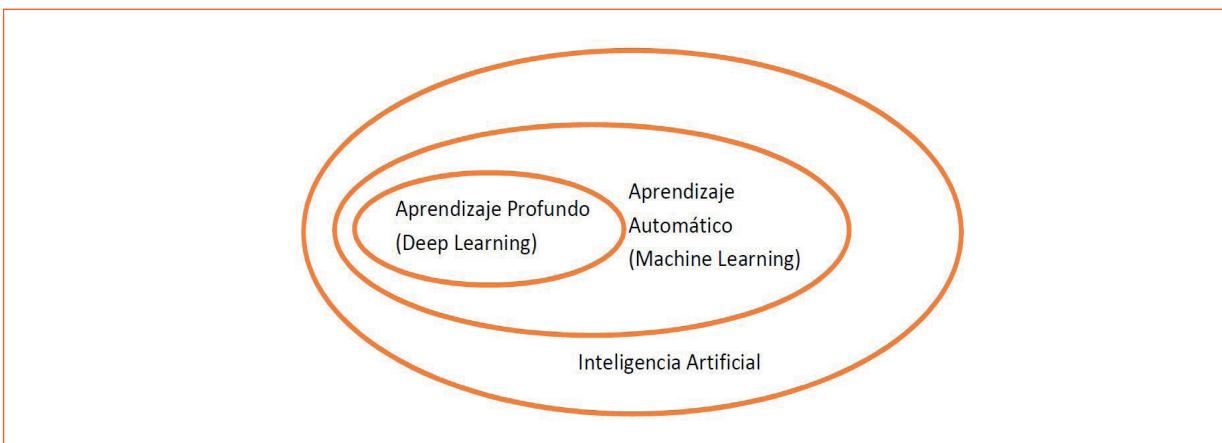


Figura 29. Áreas de mayor éxito de la inteligencia artificial.

El aprendizaje automático no modela el aprendizaje humano, es posible prever todos los problemas desde el principio, busca dar a los programas la capacidad de adaptarse, generalizando sin tener que ser reprogramados. **El aprendizaje automático es quizás una de las áreas con mayor éxito comercial en muchas aplicaciones.** Entre las múltiples aplicaciones del aprendizaje automático tenemos que nos permiten resolver problemas de reconocimiento de patrones de los datos observados, bien sea para la toma de decisiones o para descifrar el comportamiento de los datos observados. Debido al uso del internet ha emergido una fuente inmensurable de datos. El genoma humano, por ejemplo, ha podido ser estudiado, la vasta cantidad de bases de datos en la industria química y farmacéutica, bases de datos financieras, todos disponibles hoy en día gracias a la web a una escala impensable hasta solo unos cuantos años atrás. Es así como las técnicas de aprendizaje automático han contribuido grandemente en diferentes campos de la ciencia de los datos. Por ejemplo, Google usa los datos para entender cómo responder a cada interrogante que se le hace en su buscador. El otro logro importante de Google usando aprendizaje automático es el desarrollo del automóvil de conducción autónoma. Amazon usa los datos para saber dónde y qué productos colocar en la cuenta de sus clientes, al igual que Netflix. Otra de las áreas de mayor interés ha sido el de la detección de fraudes.

En el año 2005 se llevó a cabo una competencia de robots de vehículos autónomos, los cuales debían transitar la carretera diseñada por DARPA (Agencia de Proyectos de Investigación Avanzada de Defensa) para promover los proyectos de investigación en el área de vehículos autónomos. Una carretera que tiene un recorrido de 175 millas por un terreno desértico que puede ser recorrido en menos de 10 horas, sin intervención humana. Allí participó el vehículo autónomo Stanley, desarrollado por el equipo *Stanford Racing Team*. En dicha competición recibieron 2 millones de dólares por ser el primer equipo en completar el recorrido de 132 millas en el desierto de Mohave en California. Stanley (ver Figura 30) logró terminar el recorrido en menos de 6 horas y 54 minutos con un promedio de velocidad de 19mph (Stanford Racing Team, 2006).



Figura 30. Stanley, vehículo autónomo desarrollado por Stanford Racing Team, ganador de DARPA Challenge por ser el primero en recorrer 132 millas en menos de 6 horas 54 minutos. Recuperado de: <https://cs.stanford.edu/group/roadrunner//old/index.html>

El aprendizaje automático es un campo muy amplio con infinidad de métodos y aplicaciones, con el cual se puede:

- 1. Aprender a identificar parámetros**, como por ejemplo probabilidades en redes bayesianas.
- 2. Aprender a identificar estructuras**, como la estructura de arcos de las redes bayesianas, identificando conexiones.
- 3. Identificar variables escondidas**, por ejemplo, podemos encontrar en una data de entrenamiento posibles grupos. Como en Netflix, podemos ver que hay diferentes tipos de consumidores, con diferentes gustos, de tal manera de ofrecerle al consumidor los productos de los que más interés tiene en seguir consumiendo.
- 4. Conocer el comportamiento de los datos**, es acá donde podemos clasificar los métodos de aprendizaje supervisado, no supervisado y por reforzamiento.
- 5. Predecir**, diagnosticar procesos, sistemas, resumir artículos, entre muchas otras aplicaciones.

El aprendizaje automático puede actuar de forma pasiva si se usa por ejemplo un agente observador que no tenga impacto sobre la data. De manera activa, por ejemplo, de forma en línea, mientras la data se está generando o fuera de línea, donde el aprendizaje ocurre una vez que la data es generada.

Hay diferentes tipos de salida en los algoritmos de aprendizaje automático; por ejemplo:

- **Clasificación:** se trata de crear grupos de ver si un elemento pertenece o no a un grupo definido.
- **Regresión:** es continua; permite predecir valores.
- **Generativos:** buscan modelar los datos lo más general posible.
- **Discriminativos:** buscan distinguir los datos.

A decir verdad, tomó mucho tiempo llegar a definir estos términos en el área de la IA y generar los diferentes métodos que hoy en día son conocidos, usados y aplicados. Actualmente se cuenta con herramientas de desarrollo, disponibles, a mano de quienes quieran incursionar en el área. Esto es increíble, porque al mismo tiempo permite que se desarrollen nuevas ideas y las formas de uso incrementen.

El aprendizaje automático es una importante área de la inteligencia artificial, cuyo uso puede tener un gran impacto en la sociedad actual y futura. Cada vez se va haciendo más necesario su uso para el manejo de los datos, dada la inmensurable cantidad de datos digitalizados que se tienen y facilitan su almacenamiento y uso. Más adelante se explicarán los conceptos básicos para entender y aplicar dichas técnicas. Veremos las técnicas de aprendizaje supervisado, no supervisado y aprendizaje por refuerzo. Las tareas principales del aprendizaje automático son la clasificación, los ajustes de modelo usando métodos de regresión y la agrupación de los datos. Los dos primeros métodos (clasificación y regresión) pertenecen al grupo de métodos de aprendizaje supervisado, mientras que los métodos de agrupación pertenecen a los métodos de aprendizaje no supervisado (ver Figura 31).

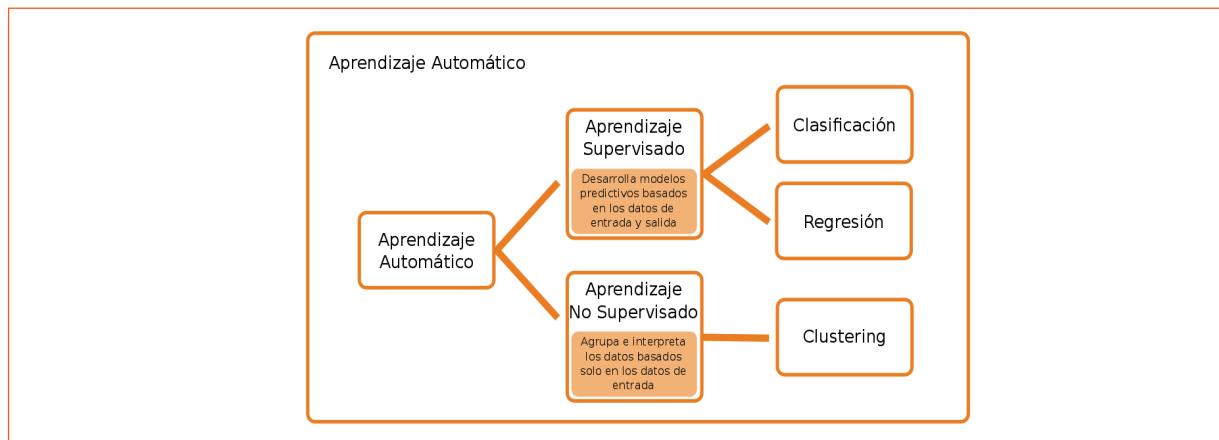


Figura 31. Introducción al aprendizaje de máquina. Recuperado de <https://www.mathworks.com/help/stats/machine-learning-in-matlab.html>

8.1. Preparación de los datos

El primer paso para trabajar con cualquier técnica de aprendizaje automático es la preparación de los datos. En dicho proceso de captura de los datos, estos deben ser correctamente elegidos y limpiados. **Una vez que se tienen los datos limpios y anotados se definen los modelos y se entrena los algoritmos.** A mayor cantidad de datos se logra obtener una mejor generalización del modelo, siempre y cuando la distribución de las instancias sea lo más cercano a la realidad.

Según el Profesor Ignacio Gómez dice en su curso de BigData (Gómez, 2019) "Siempre es mejor la calidad de los datos que la calidad de los algoritmos. Los algoritmos son más fáciles de crear de lo que parece". El mayor reto en la preparación de los datos puede ser la diversidad de los mismos. Es posible que se tenga una variedad de tipos de datos provenientes de diferentes fuentes que se deba saber manejar para poder aprovechar al máximo la contribución de dichos datos a la estimación del modelo.

Una vez recolectada la data es posible que sea necesario pre-procesarlas, con el fin de preparar los datos de entrada al modelo. Este proceso aún sufre la falta de herramientas de filtrado y extracción de características que faciliten la limpieza y transformación de los datos.

8.2. Aprendizaje supervisado

Las técnicas de aprendizaje automático supervisado tienen como entrada un conjunto de instancias, ejemplos, con sus etiquetas asociadas. El objetivo es construir un modelo capaz de predecir la etiqueta de nuevas instancias a partir del conjunto de entrenamiento ya etiquetado de los datos. Los métodos de aprendizaje más comunes son los de **clasificación y regresión**. Algunas aplicaciones ejemplos son:

1. Predecir el afluente de pasajeros que viajaron en alguna fecha festiva.
2. A partir de la información de un bien determinar su precio.
3. Dada una imagen determinar de qué se trata, de qué animal, de qué especie, etc.
4. Determinar si un mensaje es *spam*.
5. Detectar las *fake news* en las redes sociales.

Los pasos para construir un método de aprendizaje automático supervisado son (ver Figura 32):

1. Coleccionar los datos y dividirlos en datos de entrenamiento y datos de prueba. Los datos de prueba no deben estar entre los datos de entrenamiento. Existen varias técnicas que se usan para evaluar los sistemas de aprendizaje jugando o variando los conjuntos de datos de entrenamiento y de prueba.
2. Extraer las características de los datos y etiquetarlas (esto en los casos de aprendizaje supervisado).
3. Entrenar la data y crear el modelo.
4. Tomar los datos de prueba y probar el modelo.
5. Predecir los resultados o inferir sobre el comportamiento de los datos (para el caso del aprendizaje no supervisado).

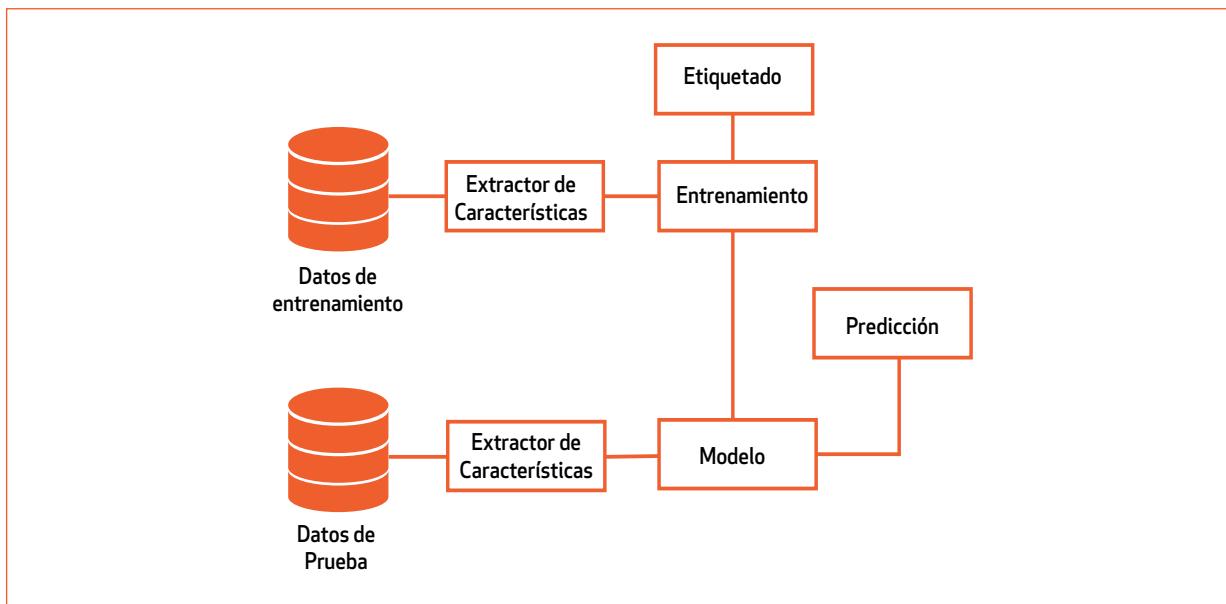


Figura 32. Pasos de los métodos de aprendizaje automático.

Más formalmente, en las técnicas de aprendizaje supervisado se define un vector de características por cada muestra usada como data de entrenamiento y cada vector de característica ($\mathbf{X}_1, \mathbf{X}_2, \mathbf{X}_3, \dots, \mathbf{X}_n$) es etiquetado (\mathbf{Y}). Sea:

$$\mathbf{X}_1, \mathbf{X}_2, \mathbf{X}_3, \dots, \mathbf{X}_n \rightarrow \mathbf{Y}$$

Un ejemplo pudiera ser un vector de característica de una agencia de crédito, cada valor X_i es una característica; por ejemplo \mathbf{X}_1 puede indicar el salario de la persona, \mathbf{X}_2 si la persona ha incumplido anteriormente con algún crédito, y así sucesivamente cada \mathbf{X}_i representa una variable característica de los datos. Sea \mathbf{Y} el valor a predecir si la persona va a incumplir con el crédito o no. Se observa el pasado de los datos donde la agencia de crédito ha tomado las muestras, con los datos de la persona y las ocurrencias de incumplimiento o no. Lo que se desea producir es una función que nos permita predecir los futuros clientes, así que cuando un nuevo cliente venga a solicitar un crédito podemos predecir según sus datos si es un potencial cliente para el crédito o no, de tal manera que se pueda obtener una relación funcional entre las características $\mathbf{X}_1, \mathbf{X}_2, \mathbf{X}_3, \dots, \mathbf{X}_n$ y \mathbf{Y} . Se puede aplicar lo mismo con las imágenes, donde los vectores característicos pueden ser píxeles de las imágenes, o características de cosas encontradas en las imágenes, y la \mathbf{Y} si la imagen contiene o no cierto objeto. En Aprendizaje supervisado se tienen varias muestras, por ejemplo:

$$\left[\begin{array}{l} \mathbf{X}_{11}, \mathbf{X}_{12}, \mathbf{X}_{13}, \dots, \mathbf{X}_{1n} \rightarrow \mathbf{Y}_1 \\ \mathbf{X}_{21}, \mathbf{X}_{22}, \mathbf{X}_{23}, \dots, \mathbf{X}_{2n} \rightarrow \mathbf{Y}_2 \\ \mathbf{X}_{m1}, \mathbf{X}_{m2}, \mathbf{X}_{m3}, \dots, \mathbf{X}_{mn} \rightarrow \mathbf{Y}_m \end{array} \right] \rightarrow \mathbf{data}$$

Sea cada vector característico \mathbf{X}_m podemos definir la función $f(\mathbf{X}_m) \rightarrow \mathbf{Y}$, donde \mathbf{Y} sería el valor más cercano posible al valor real. Sin embargo, esto no siempre es posible, aunque puede aproximarse a un valor aceptable, o tolerable con un cierto margen de error de entrenamiento. El objetivo del aprendizaje automático es estimar la función f , lo más cerca posible de su hipótesis h . Una vez identificada se

puede seguir usando para cualquier otro conjunto de datos característicos que no fueron partes del conjunto de entrenamiento y producir un valor de salida predictivo con una asertividad aceptable.

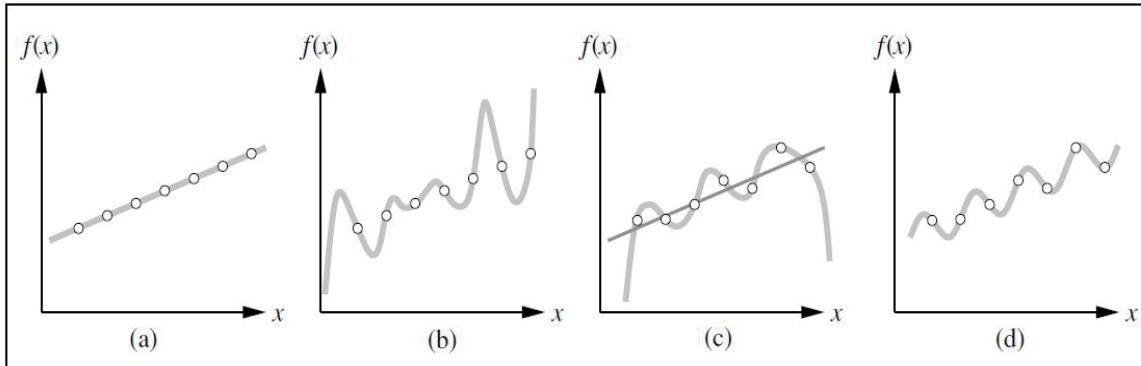


Figura 33. Ejemplos de pares $(x, f(x))$ y una hipótesis lineal consistente. (b) Un polinomio de grado 7 como hipótesis consistente para el mismo conjunto de datos. (c) Un conjunto de datos diferentes que admite un polinomio de grado 6 que ajusta de forma exacta o un encaje lineal aproximado. (d) Un encaje sinusoidal sencillo para el mismo conjunto de datos. Recuperado de Russell y Norvig (2010).

Ahora cómo saber cuándo se ha conseguido la función que mejor se ajuste a los datos, de tal manera que se ajuste a los datos y sea eficiente. Como se observa en la Figura 33, los datos pueden aproximarse a varias funciones, unas más complejas que las otras, donde la más simple es la lineal (Figura 33 a). Generalmente se recomienda escoger la función más simple que se ajuste, y que, aunque no sea exactamente consistente permita igualmente hacer predicciones razonables. En algunos casos sucede que no se dieron en totalidad las muestras que debían darse para obtener un mejor ajuste. Así es como se dice que un aprendizaje es realizable si en el espacio de hipótesis se tiene la función verdadera, de lo contrario no es realizable. No siempre el aprendizaje es realizable, pero es complicado aún saber cuál puede ser esa función verdadera.

Para escoger la mejor función que aproxime los valores tomamos en cuenta lo que se le denomina el principio lógico de Ockham (Audi, 1995) o también llamado la navaja de Ockham, el principio de economía o principio de parsimonia, el cual dice "Mientras más sencillo mejor". Básicamente, que es mejor no abarcar mucho más de lo que es absolutamente necesario. Los métodos de Aprendizaje automático requieren de un conjunto de datos de entrenamiento, pero si usamos muy pocos es imposible estimar la función y caemos en lo que se denomina **subestimación**. Si usamos muchos datos es posible que se caiga en un error de **generalización** y la función de predicción es sobreajustada (*overfitting error*). La mejor complejidad se obtiene donde el error de generalización es mínimo (optimo) (ver Figura 34).

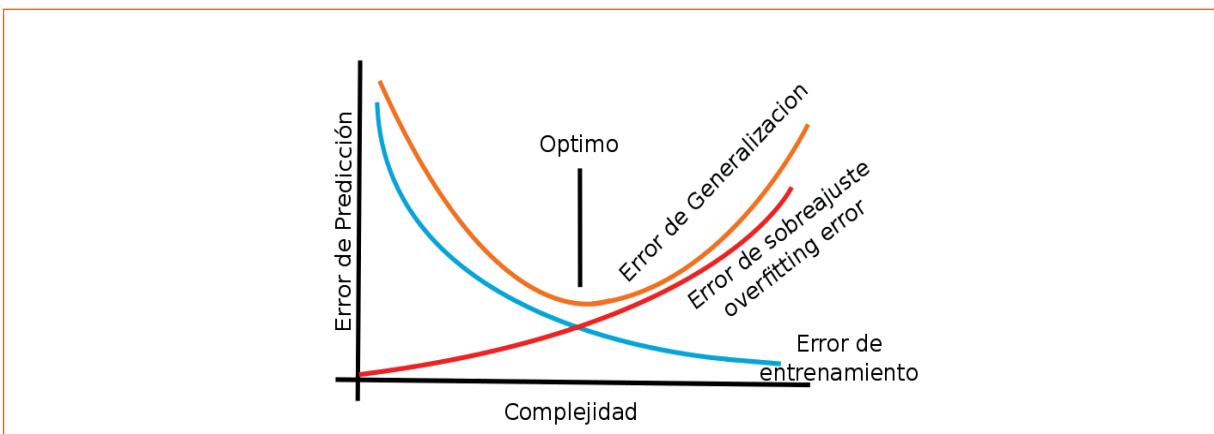


Figura 34. Gráfico de complejidad de la generalización del modelo a ajustar.

8.3. Aprendizaje no supervisado

En el aprendizaje no supervisado, a diferencia del aprendizaje supervisado, los datos no están previamente etiquetados. Al igual que en las técnicas de aprendizaje supervisado se tiene como entrada un conjunto de datos o instancias. El objetivo, sin embargo, es conseguir grupos de instancias similares. En este caso, al ser datos no identificados *a priori*, no se tienen los expertos que especifiquen los grupos que se deben buscar. El aprendizaje no supervisado permite encontrar patrones escondidos o estructuras intrínsecas en los datos observados. Es mayormente usado para inferir sobre los datos a partir de la consistencia de los datos sin ser etiquetados *a priori*. Los métodos de agrupamiento o *clustering* son los más usados de las técnicas de aprendizaje no supervisado. Son usados para explorar los datos, encontrar patrones escondidos o agruparlos. Entre sus aplicaciones está el análisis de genes, la investigación de mercado y el reconocimiento de objetos. Entre los tipos de métodos de aprendizaje no supervisado, están:

1. Agrupación (*clustering*).
2. Agrupación jerárquica.
3. Reducción de dimensionalidad.

Entre las aplicaciones más comunes, podemos encontrar:

1. Agrupar libros por títulos.
2. Agrupar imágenes por distribución de colores.
3. Agrupar palabras por contexto.

8.4. Clustering

Las técnicas de *clustering*, o agrupamientos, son métodos utilizados para encontrar patrones escondidos de estructuras intrínsecas en la data. Es la técnica más común de aprendizaje no

supervisado, es utilizado justamente para analizar los datos exploratorios y encontrar patrones o grupos de datos con similitudes.

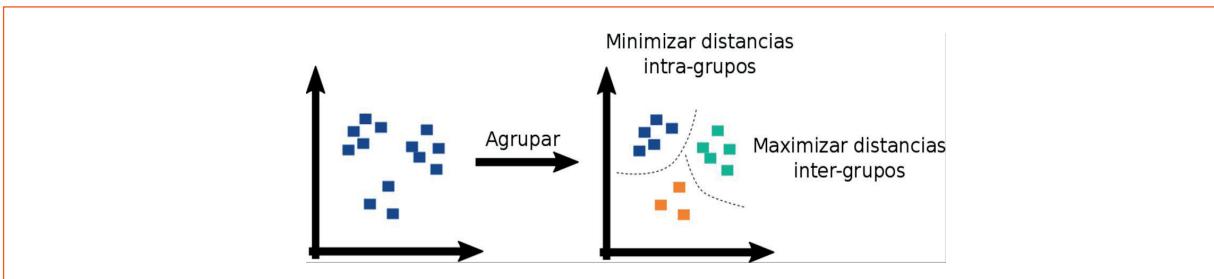


Figura 35. Ilustración del resultado de la técnica de agrupamiento o *clustering* de los datos.

El agrupamiento se logra minimizando las distancias intragrupales y maximizando las distancias intergrupales. La idea es separar grupos distintivos de datos (ver Figura 35). Estas técnicas son muy utilizadas en segmentación de imágenes, reconocimiento de objetos, entre otras.

8.5. Datos de entrenamiento vs datos de prueba

Generalmente se tiene un conjunto de datos finitos, tanto para el entrenamiento como para las pruebas. Una vez preparados los datos, tal y como se explica en la sección de aprendizaje supervisado, mientras más datos se tengan mejor se puede aproximar al modelo y se obtiene una mejor generalización de los datos. Los datos de entrenamientos deben ser usados solo para el entrenamiento del modelo, y los datos de pruebas no pueden ser parte de los datos de entrenamiento, de lo contrario se producirá un sesgo en los datos y no se obtendrá un modelo real de aproximación a los datos. Es posible particionar los datos en dos grupos. Un primer grupo escogido de manera aleatoria que corresponda al 2/3 de los datos, puede ser utilizado como data de entrenamiento, mientras el otro tercio de la data puede ser utilizado para validar la precisión en la estimación del clasificador. Otra manera de particionar los datos en tres grupos en el que el 80% pertenezca a los datos de entrenamiento, un 10% a datos usados para la validación del método y un 10% restante para pruebas (ver Figura 36).



Figura 36. Distribución de los datos para ser usados en el entrenamiento y prueba de los sistemas de aprendizaje no supervisado.

Los datos para la validación pueden ser validados usando lo que se denomina la validación cruzada.

8.5.1. Validación cruzada *k-fold* (*K-fold cross-validation*)

Esta técnica de validación se realiza dividiendo los datos en ***k*** conjuntos de datos disjuntos de igual tamaño, donde cada conjunto tiene la misma distribución. Se entrena el clasificador con cada conjunto de datos, es decir *k* veces. El error estimado en cada ejecución es usado para determinar la media del error. De esta forma se valida el clasificador.

En un clasificador la ratio de error define el porcentaje de hacer clasificaciones incorrectas. Y la precisión viene dada por el porcentaje de clasificaciones correctas, es decir ***E=error***, y ***Precision = 1 - error***.

8.5.2. Comparación de métodos de aprendizaje

Una forma de comparar y evaluar la eficiencia de los métodos de aprendizaje y verificar que nuestro modelo se ajusta a los datos, es utilizando lo que se define como la matriz de confusión o las curvas ROC.

8.5.3. Matriz de confusión

La matriz de confusión es una herramienta de evaluación del desempeño de los algoritmos de aprendizaje supervisado, es muy utilizado para evaluar los clasificadores. Se construye una matriz cuadrada, donde se cruzan los datos entre las predicciones y las instancias reales. Las columnas son las predicciones y las filas suman las instancias reales. Los clasificadores son capaces de predecir a qué clase pertenece una instancia dada. Una matriz de confusión ayuda a verificar la precisión y exactitud con que el modelo se aproxima a la correcta clasificación. El caso más sencillo es cuando se trabaja con un clasificador binario, donde se tienen dos clases, positivo o negativo, acierto o no acierto.

Tabla 7
Matriz de Confusión

	Determinístico	Estocástico
Completemente Observable	A*, Primero en profundidad, Primero en Anchura	Proceso de Decisión de Markov
Parcialmente Observable		Proceso de Decisión de Markov Parcialmente Observable

- **Negativos reales (verdaderos negativos):** el clasificador arroja negativo y la instancia es negativa.
- **Positivos falsos (falsos positivos):** El clasificador arroja positivo cuando el valor real es negativo.
- **Falsos negativos (falsos negativos):** El clasificador dice que la instancia es positiva, cuando es falsa.
- **Positivos reales (verdaderos positivos):** El clasificador dice que la instancia es positiva y acierta.

Los valores de la diagonal en la matriz de confusión se tienen el total de las predicciones que fueron acertadas, bien sea para los casos negativos como los positivos. Mientras que los otros valores son los casos de falla del clasificador. Por si solo estos valores no me indican la asertividad del modelo, para ello se realizan los cálculos de la precisión y exactitud o sensibilidad y especificidad.

Precisión y Exactitud

La precisión viene dada por:

$$\text{Precision} = \frac{\text{Predicciones correctas}}{\text{Total de predicciones}} = \frac{VN+VP}{(VN+FP+FN+VP)}$$

La precisión mide lo disperso que es el modelo: a menor precisión mayor será la exactitud.

Mientras que la exactitud:

$$\text{Exactitud} = \frac{VP}{FP+VP}$$

Mide la relación entre las predicciones positivas reales y las predicciones positivas acertadas y no acertadas.

Sensibilidad y Especificidad

Estos valores de sensibilidad y especificidad permiten evaluar la capacidad del clasificador para discriminar los casos positivos de los negativos. **La sensibilidad mide la tasa de los verdaderos positivos y la especificidad la tasa de los verdaderos negativos.**

$$\text{Sensibilidad} = \frac{VP}{FN+VP}$$

Y la especificidad:

$$\text{Especificidad} = \frac{VN}{VN+FP}$$

Dependiendo del caso, a veces es importante buscar que la sensibilidad sea alta, o a veces se toma la especificidad, depende del costo del error. Si tiene un costo alto de error es conveniente usar la sensibilidad como métrica. Hasta ahora evaluamos las métricas usadas en un clasificador binario, pero si es un clasificador o discriminador de grupos, la matriz de confusión se vuelve más compleja.

8.5.4.Curvas ROC

La curva ROC por su acrónimo en inglés *Reveiver Operating Characteristic*, es una gráfica que se crea a partir de los datos de sensibilidad vs especificidad. Es básicamente una herramienta de análisis de los clasificadores utilizada para seleccionar los mejores y descartar los no tan óptimos.

La curva ROC (ver Figura 37) se crea a partir de los valores de **sensibilidad** y **1-especificidad** en la medida en que la curva se aproxima a los valores cercanos a 1, se tiene un clasificador perfecto, muy bueno o mejor. En la medida en que se acerca a la diagonal, 50% de sensibilidad y especificidad, el clasificador tiene una respuesta aleatoria, no es un buen clasificador. Recordemos que la sensibilidad mide la tasa de verdaderos positivos, es decir la tasa en las que el clasificador detecta como verdaderos los casos de clasificación correcta.

Esta curva ROC (ver Figura 37) nos permite entonces poder analizar y evaluar la eficacia del clasificador. Se espera de un buen clasificador tener como resultado una curva ROC cercana al valor de clasificación perfecta en la que la tasa de verdaderos positivos sea lo más cercano a 1.

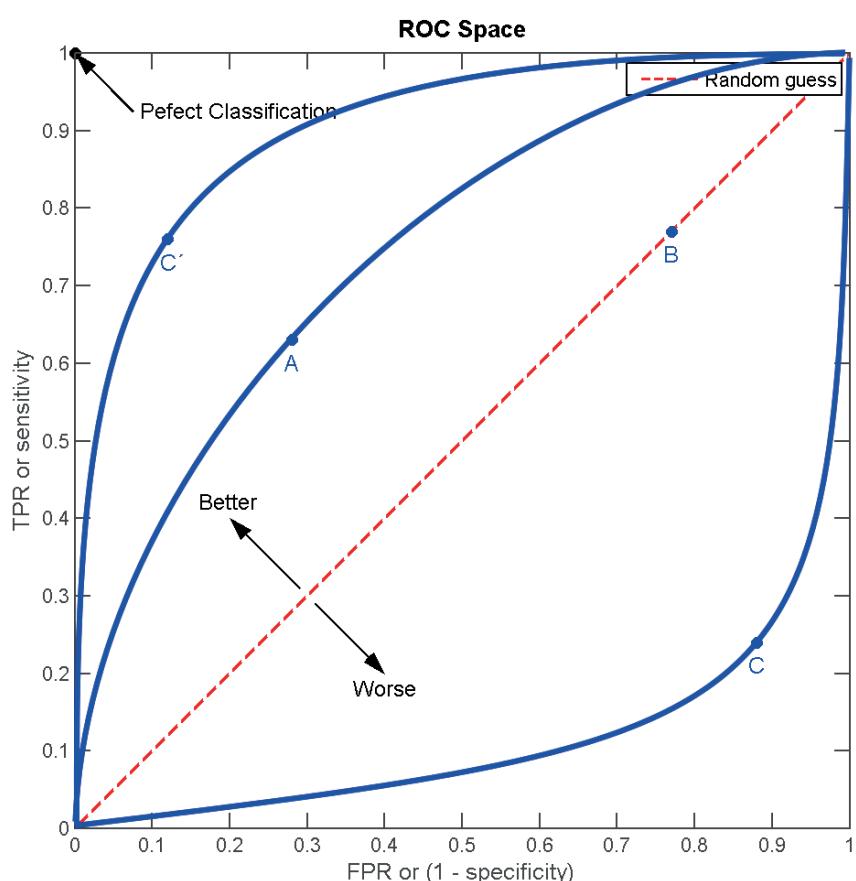


Figura 37. Curva ROC, mostrando la tendencia de varios tipos de resultados de la curva ROC. Recuperado de <http://www.ericmelillanca.cl/content/evaluaci-n-modelos-clasificaci-n-matriz-confusi-n-y-curva-roc>

8.6. Seleccionando el algoritmo correcto de aprendizaje

Dentro de los métodos de aprendizaje automático podemos mencionar varios desde los más sencillos a los más complejos. Una buena clasificación de métodos la vemos en la Figura 38. Escoger el método de aprendizaje automático que mejor se ajuste a los datos y al problema dependerá por supuesto de la información que se tenga de los datos y lo que se quiera lograr con ellos. Sin embargo, no hay una receta para su selección, en la mayoría de los casos se hace ensayo y error, si no funciona con un método se intenta con otro, hasta encontrar el método que mejor resultados arroje. Pero lo más

importante a tomar en cuenta es que el conjunto de datos de entrenamiento debe ser correctos. Los modelos flexibles tienden a sobrecargar los datos al modelar pequeñas variaciones que podrían ser ruido. Los modelos simples son más fáciles de interpretar, pero pueden tener menor precisión. Escoger un método de aprendizaje involucra poner en la balanza la velocidad, precisión y complejidad. Hasta hoy en día en la mayoría de los casos se trata de un problema de ensayo y error.

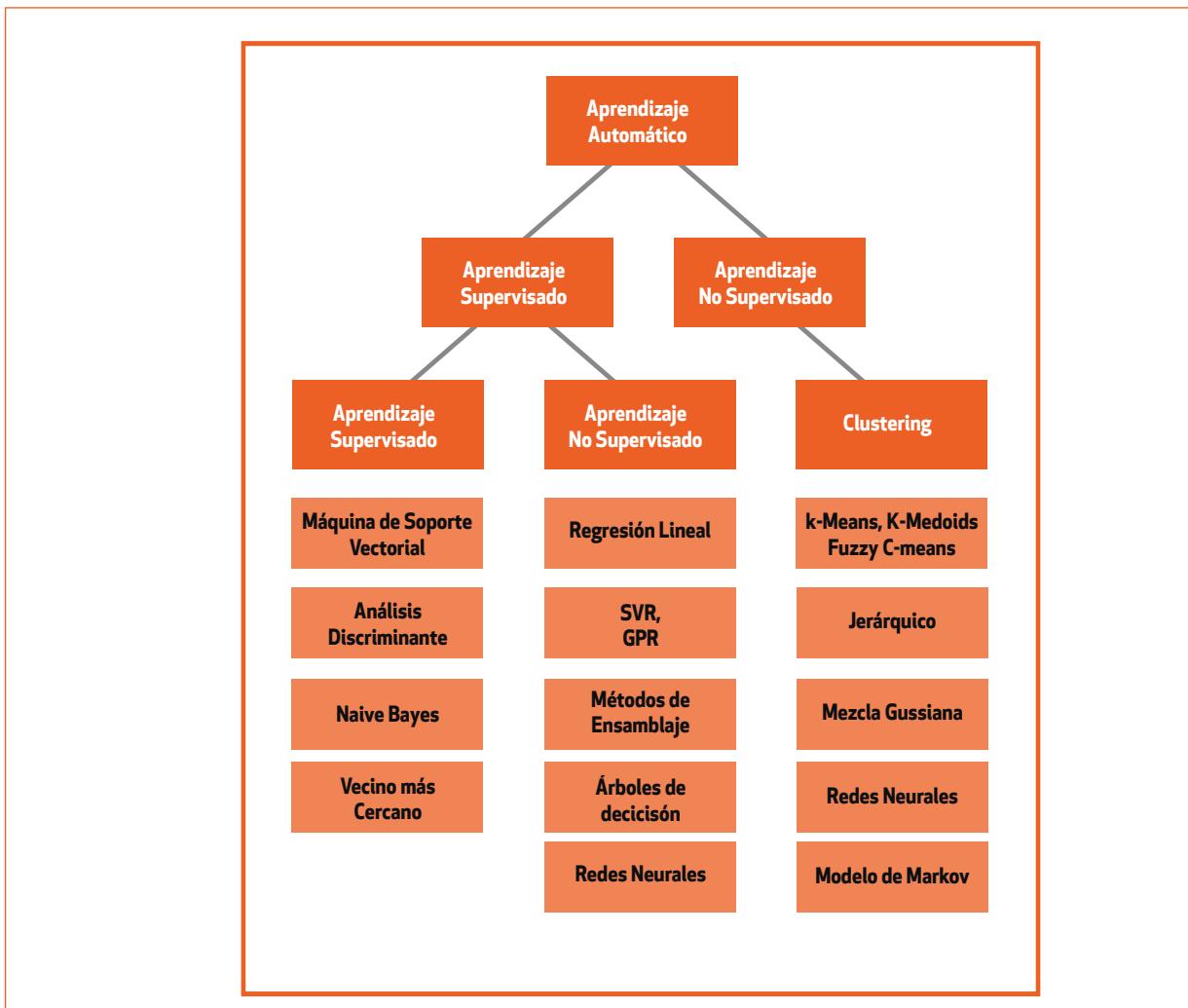


Figura 38. Introducción al aprendizaje de máquina. Adaptado de <https://www.mathworks.com/help/stats/machine-learning-in-matlab.html>

8.6.1.Herramientas que te ayudan a escoger un método de aprendizaje

Matlab® es una herramienta robusta poderosa muy recomendada para el análisis de datos, para implementar los algoritmos de procesamiento de datos (señales, imágenes, vectoriales, etc.) y crear modelos matemáticos. La colaboración con varios expertos ha permitido el desarrollo de aplicaciones y métodos de aprendizaje automático. Matlab® cuenta con una aplicación que permite comparar los resultados usando diferentes clasificadores (Classification Learner App), ver Figura 39. De tal manera que se pueda entrenar la data y obtener el modelo que mejor se ajuste y el calificador o método de aprendizaje que arroje mejores resultados. La aplicación le permite explorar el aprendizaje automático supervisado de forma interactiva utilizando varios clasificadores. Dicha herramienta permite entrenar automáticamente diferentes modelos ayudando a escoger el mejor modelo que se acomode a los datos.

Incluye los modelos de árboles de decisión, análisis discriminante, máquinas de soporte vectorial, regresión, vecinos más cercanos y clasificación por conjuntos. Esta aplicación de Matlab® permite además comparar los modelos usando las curvas ROC o la matriz de confusión.

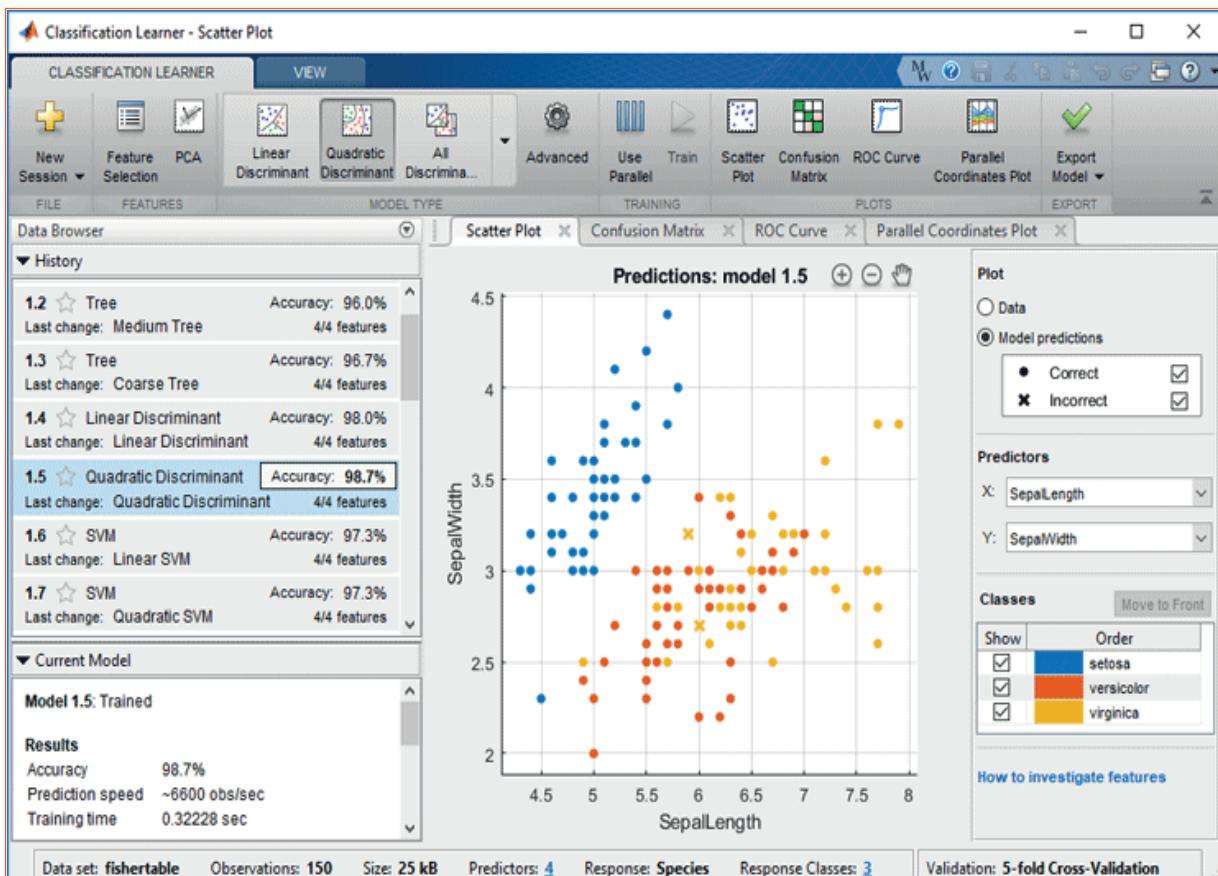


Figura 39. Classification Learner App. Fuente: <https://www.mathworks.com/help/stats/machine-learning-in-matlab.html>

8.7. Aprendizaje por refuerzo

El aprendizaje por refuerzo es el conjunto de técnicas en las que se aplica el reforzamiento del aprendizaje al agente inteligente mediante la aplicación de recompensas o castigo, cuando la acción tomada es buena o mala. En el aprendizaje por refuerzo se tiene un agente, quien ejecuta una acción, el resultado de la cual se interpreta y el agente recibe entonces el estado, y una recompensa (positiva o negativa, según el caso). A diferencia de otras técnicas de aprendizaje, en el aprendizaje por refuerzo no se tienen los pares entradas y salidas etiquetadas; se tiene un agente que tiene que adquirir experiencia útil acerca de los estados, acciones y recompensas, de forma pasiva o activa para poder ejecutar una próxima acción. La evaluación del aprendizaje en este caso se realiza de manera concurrente con el aprendizaje.

Un programa para jugar a damas fue la primera aplicación de aprendizaje por refuerzo implementada. Sin embargo, en esta aplicación la recompensa no se usó en los estados terminales, así que de la misma forma como podía ganar, asimismo podía perder. Luego realizaron varias modificaciones

que les permitió mejorar su técnica para ganar. Aplicaciones muchas, las más recientes podemos mencionar:

- Watson (IBM) - campeón en el juego de Jeopardy (2011).
- Atari 2600 - Aprendió como jugar 46 video juegos, superando en 29 a humanos (Steven 2001).
- AlphaGo - campeón en el juego de Go en 2015. AlphaGo fue desarrollado por el equipo de Google DeepMind.
- Múltiples aplicaciones en robótica.

Generalmente en los juegos es donde mayormente se aplican estas técnicas de aprendizaje por reforzamiento. El objetivo principal es crear un agente cuyas acciones le lleven a acumular el mayor número de recompensas positivas, es decir de éxito. El objetivo del agente es encontrar la forma que maximice a largo plazo el refuerzo acumulado. Generalmente se trabaja en ambientes:

- **No determinísticos:** es válido tomar la misma acción en el mismo estado y obtener resultados diferentes.
- **Estacionarios:** la probabilidad de cambiar de estado es baja o cambian de manera muy lenta.

Se definen dos tipos de aprendizaje por refuerzo:

- **Aprendizaje pasivo:** el agente tiene una política fija de actuar, y requiere aprender del entorno y sus estados.
- **Aprendizaje activo:** el agente debe aprender que hacer en cada paso, no tiene una política fija de actuación.

8.8. Aprendizaje profundo

El aprendizaje profundo es un campo del aprendizaje automático, utiliza lo que se denomina técnicas de redes neuronales profundas. Procura un aprendizaje de múltiples niveles de representación incrementando su complejidad y abstracción. Para implementar los métodos de aprendizaje profundo se utilizan grandes cantidades de datos, por lo que es necesario programar usando programación en *hardware GPU*.

A diferencia del modelo de aprendizaje automático, el aprendizaje profundo no requiere del proceso de extracción de características (ver Figura 40). **El modelo por sí solo aprende a distinguir los distintos elementos, clases.** El modelo **aprende a extraer las características de los objetos.** Ambas técnicas aprenden por ejemplo a identificar el objeto a extraer, bien sea en imágenes o en un conjunto de datos, y a extraer conceptos o variables. La diferencia está en el modo de ejecución.

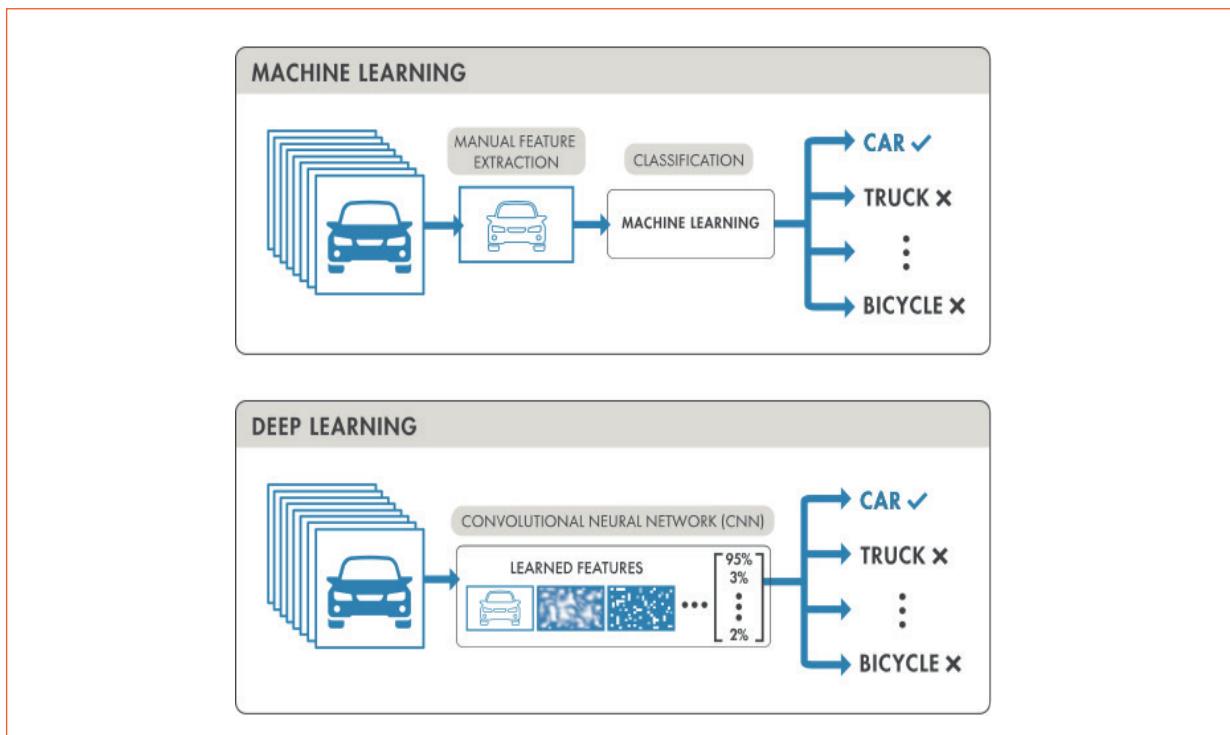


Figura 40. Ejemplo ilustrado para mostrar la diferencia entre aprendizaje automático (*Machine Learning*) y aprendizaje profundo (*Deep Learning*). Recuperado de <https://www.coursera.org/learn/ai-for-everyone>

En el aprendizaje profundo (**Deep Learning**) se utilizan mayormente las redes neuronales convolucionales (**CNN** por sus siglas en inglés, *Convolution Neural Networks*). Generalmente existen dos formas de aplicar aprendizaje profundo en los datos:

- **Entrenando el modelo desde cero**, requiere de grandes cantidades de datos, correctamente etiquetadas y diseñar una arquitectura de red que aprenda a distinguir las características y cree el modelo. Para ello, se requiere una cantidad grande de datos, y jugar un poco con la arquitectura de red y los pesos en los nodos. Se requiere de realizar varias pruebas.
- **Usando ya una red entrenada con otro conjunto de datos y aplicarlos al problema particular**, haciendo lo que se le ha denominado **transferencia de aprendizaje**. En el cual se tiene una red entrenada, probada y analizada. Se le proporcionan nuevos datos, referentes a los nuevos datos del problema a resolver y se le proporcionan datos adicionales o nuevos.

Esta forma de aplicación del aprendizaje profundo es menos costosa en tiempo, requiere menos tiempo pues ya se está trabajando con una red entrenada. Hoy en día existen varias redes ya entrenadas disponibles que se pueden utilizar para entrenar un nuevo conjunto de datos aplicando transferencia de aprendizaje; de las conocidas podemos mencionar a **AlexNet**, **GoogleLetNet**, etc. Si *a priori* se conocen las características de los datos, quizás aplicar algún método de aprendizaje automático resulta más útil o eficiente en cuanto a tiempo y cantidad de datos requeridos que si aplicamos aprendizaje profundo y de igual manera se obtienen buenos resultados. El principal argumento para utilizar aprendizaje profundo es si se dispone primero de un buen equipo de *hardware* que permita procesar tal cantidad de datos y por su puesto se cuenta con grandes cantidades de datos (ver Figura 41).

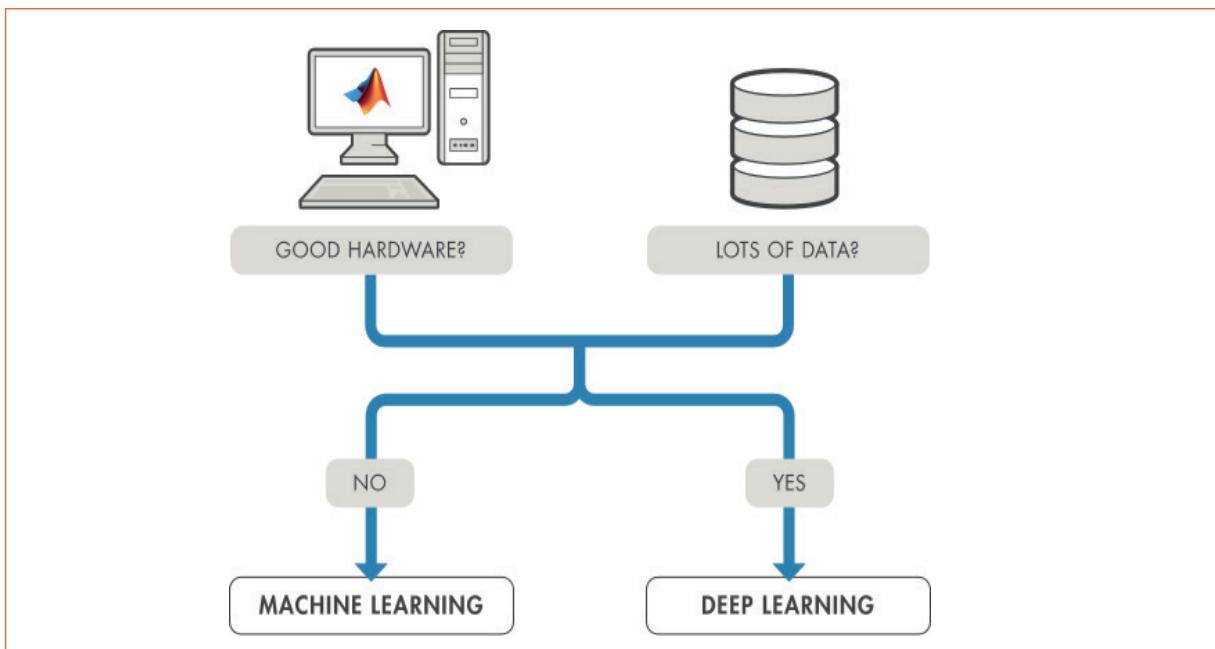


Figura 41. Ejemplo ilustrado para mostrar la diferencia entre aprendizaje automático (*Machine Learning*) y aprendizaje profundo (*Deep Learning*). Recuperado de <https://www.coursera.org/learn/ai-for-everyone>

9. Modelo de Markov

En la sección de inferencia probabilística hablamos sobre la construcción de las cadenas de Markov. Estas son un caso particular de los procesos de Markov, en los que se tiene un número finito de estados y con tiempos discretos. **Los procesos de Markov, son más generales, los estados pueden ser numerales o no, y el tiempo puede ser discreto o continuo.**

Un ejemplo de un proceso de Markov discreto puede ser por ejemplo en meteorología, generando secuencias de días con lluvia o sin lluvia. La probabilidad de lluvia o no lluvia dependerá de lo que suceda el día anterior. Se puede concluir que un proceso de Markov es **un proceso estocástico donde la probabilidad condicional de futuros estados depende del estado actual y no de los estados pasados.**

En robótica se utilizan extensivamente los modelos markovianos ocultos, así que es un tema fundamental para poder desarrollar técnicas de IA en el área robótica. En términos generales los procesos de Markov son utilizados en un sin fin de aplicaciones; entre ellas, en economía, en finanzas, en medicina y en robótica entre muchas otras. Un proceso markoviano oculto o HMM (*Hidden Model Markov*) es un proceso de Markov con parámetros desconocidos, aleatorios e influyen en el comportamiento de las observaciones del presente. Dicha característica permite manejar la incertidumbre o verosimilitud. **Los procesos de aprendizaje buscan maximizar la verosimilitud.**

Un ejemplo de un proceso de Markov es el movimiento browniano, un movimiento aleatorio que realizan las partículas suspendidas en un fluido (un líquido o un gas) como resultado de su colisión con los átomos

o moléculas de movimiento rápido en el gas o el líquido. El movimiento browniano es un proceso de Márkov ya que el desplazamiento de la partícula no depende de sus desplazamientos pasados.

9.1. Modelo oculto de Markov (HMM)

Un modelo oculto de Markov es un modelo estadístico en el que se asume que el sistema que está siendo modelado es un proceso de Markov con estados no observados (ocultos). Puede ser considerado como una red bayesiana dinámica simple. Estos modelos ocultos de Markov son muy utilizados en aplicaciones para el reconocimiento del habla, de escritura manual, de gestos, de interpretaciones gestuales, o en bioinformática. Un ejemplo práctico es en aplicaciones de traducción mediante reconocimiento de voz, palabras habladas a texto.

En los modelos de Markov, los estados son observables y las probabilidades de transición entre estados son los únicos parámetros. En el modelo oculto de Markov hay estados no observables, solo las variables, y estas son influidas por el estado. Cada estado tiene una distribución sobre los posibles elementos de salida, y esta secuencia de elementos generada proporciona entonces cierta información acerca de la secuencia de estados (Rabiner, 1989).

Este método es utilizado generalmente para analizar o predecir series temporales. Estas aplicaciones incluyen:

- a. Robótica.
- b. Medicina.
- c. Finanzas.
- d. Reconocimiento de voz y tecnologías del lenguaje.
- e. Entre muchos otros...

Siempre en las series de tiempo que involucren, ruido, sensores o incertidumbre, HMM es el método a seleccionar. La esencia de HMM es que está caracterizado por la siguiente red bayesiana, en la que se tiene una secuencia de estados en el tiempo.

$$S_1 \rightarrow S_2 \rightarrow S_3 \rightarrow \cdots \rightarrow S_n$$

Y cada estado S_i depende solo del estado anterior S_{i-1} en la red bayesiana. Cada estado emite una medida Z_i (ver Figura 42).

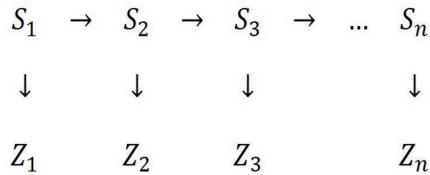


Figura 42. Representación de los estados y de las variables observables que influyen en el resultado de cada estado.

Dicha red es el núcleo de los modelos ocultos de Markov (HMM). Varios filtros probabilísticos usan estos modelos; tales como el filtro Kalman, el filtro de partículas y muchos otros. Si observamos la secuencia $S_1 \rightarrow S_2 \rightarrow S_3 \rightarrow \cdots \rightarrow S_n$ es justamente una representación de la cadena de Markov, en la que el estado S_i depende solo del estado S_{i-1} y no del estado anterior al S_{i-2} . Cuando hablamos de los modelos ocultos de Markov, hablamos de la existencia de ciertas variables Z_i entonces en vez de ver solo estados, observamos también algunas mediciones, que son llamadas mediciones observables.

9.2. Arquitectura de un modelo oculto de Markov

Sea $S = \{1, \dots, M\}$ la secuencia o cadena de Markov, el conjunto de estados, se representa la secuencia de estados como:

$$S_{1:T} \triangleq S_1 \rightarrow S_2 \rightarrow S_3 \rightarrow \cdots \rightarrow S_T$$

donde S_t es el estado en el momento de tiempo t . Se denota $s_{1:T}$ como una realización de $S_{1:T}$. Una secuencia de observaciones se denota como:

$$O_{1:T} \triangleq O_1 \rightarrow O_2 \rightarrow O_3 \rightarrow \cdots \rightarrow O_T$$

y $O_t \in Z$ donde $Z = \{z_1 \rightarrow z_2 \rightarrow z_3 \rightarrow \cdots \rightarrow z_k\}$ el conjunto de valores observables. Una observación es producida en cada estado i .

La probabilidad de transición de un estado i al estado j viene dado por:

$$a_{ij} \triangleq P(S_{t+1} = j | S_t = i)$$

donde $\sum_{j \in S \setminus \{i\}} a_{ij} = 1$ y $a_{ii} \neq 0$ y la probabilidad de emisión de observaciones del estado i viene dada por:

$$b_i(z_k) \triangleq P(o_t = z_k | S_t = i)$$

donde $\sum_k b_i(z_k) = 1$

Y la probabilidad del estado inicial $\pi_i \triangleq P(S_1 = i)$ donde $\sum_i \pi_i = 1$, finalmente, un modelo oculto de Markov esta definido por el hiperparámetro número de estados M y el conjunto de parámetros:

$$\lambda \triangleq \{a_{ij}, b_i(z_k), \pi_i\}$$

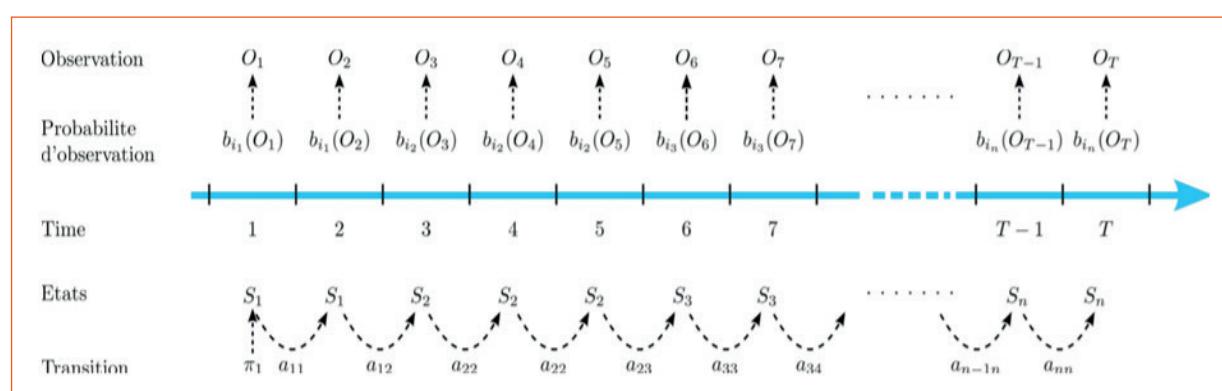


Figura 43. Ejemplo de funcionamiento de un modelo oculto de Markov. Recuperado de Altuve (2011).

En la Figura 43 hay un ejemplo general de un modelo oculto de Markov. Se observan las variables observables, la probabilidad de observación de las mismas, el tiempo, los estados y la transición entre los estados. En la Figura 44 vemos otra ilustración de un modelo oculto de Markov.

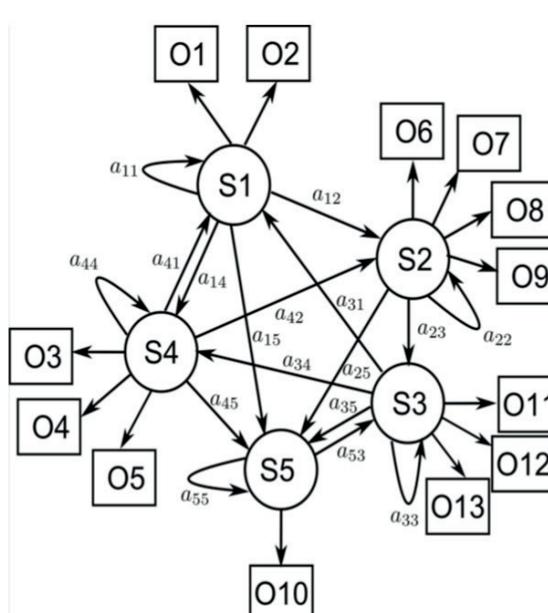


Figura 44. Otro ejemplo de un modelo oculto de Markov. Recuperado de Altuve (2011).

9.3. Estimación de los parámetros en un modelo oculto de Markov

Los parámetros del modelo son estimados, generalmente, utilizando el método de maximización de la verosimilitud. El conjunto de parámetros λ se obtiene a partir de las observaciones $O_{1:T}$. Una vez estimado λ se reestima de nuevo, a fin de que la función de verosimilitud $L(\lambda; O_{1:T}) = P(O_{1:T} | \lambda)$ aumente hasta que converja a su máximo valor, donde $P(O_{1:T} | \lambda)$ sería entonces la probabilidad de que el modelo genere la secuencia observada $O_{1:T}$ con el conjunto de parámetros λ .

Generalmente se utiliza un enfoque basado en el algoritmo maximiza-esperanza, dicho algoritmo busca iterativamente encontrar la máxima verosimilitud usando dos etapas:

1. Determinando la esperanza de la log-verosimilitud según las observaciones y los parámetros actuales:

$$Q(\lambda | \lambda^{(t)}) = E[\log L(\lambda; O_{1:T})]$$

2. Encontrar los parámetros que maximizan:

$$\lambda^{(t+1)} = \arg \max_{\lambda} Q(\lambda | \lambda^{(t)})$$

También se pueden usar el algoritmo de *forward-backward* y el algoritmo de Viterbi. Estos dos se usan a menudo, provienen de un enfoque iterativo de máxima-verosimilitud. El algoritmo *forward-*

backward considera el conjunto de observaciones para determinar $L(\lambda; O_{1:T})$ mientras que el método de Viterbi solo tiene en cuenta la ruta óptima (Levinson (1986)).

9.4. Algoritmo de Viterbi

El algoritmo de Viterbi (Forney, 1993) encuentra las secuencias de estados más probables en un modelo oculto de Markov. A partir de una observación, o mediciones de variables observables. Inicialmente se le debe dar como entrada el grafo de estados y las observaciones en el tiempo inicial.

```

function VITERBI(observations of len  $T$ , state-graph of len  $N$ ) returns best-path
    create a path probability matrix viterbi[ $N+2, T$ ]
    for each state  $s$  from 1 to  $N$  do ; initialization step
        viterbi[ $s, 1$ ]  $\leftarrow a_{0,s} * b_s(o_1)$ 
        backpointer[ $s, 1$ ]  $\leftarrow 0$ 
    for each time step  $t$  from 2 to  $T$  do ; recursion step
        for each state  $s$  from 1 to  $N$  do
             $viterbi[s, t] \leftarrow \max_{s' = 1}^N viterbi[s', t - 1] * a_{s', s} * b_s(o_t)$ 
            backpointer[ $s, t$ ]  $\leftarrow \operatorname{argmax}_{s' = 1}^N viterbi[s', t - 1] * a_{s', s}$ 
    viterbi[ $q_F, T$ ]  $\leftarrow \max_{s = 1}^N viterbi[s, T] * a_{s, q_F}$  ; termination step
    backpointer[ $q_F, T$ ]  $\leftarrow \operatorname{argmax}_{s = 1}^N viterbi[s, T] * a_{s, q_F}$  ; termination step
    return the backtrace path by following backpointers to states back in
        time from backpointer[ $q_F, T$ ]

```

Figure 9.11 Viterbi algorithm for finding optimal sequence of hidden states. Given an observation sequence and an HMM $\lambda = (A, B)$, the algorithm returns the state path through the HMM that assigns maximum likelihood to the observation sequence. Note that states 0 and q_F are non-emitting.

Figura 45. Algoritmo de Viterbi, encuentra la secuencia optima de estados en un modelo oculto de Markov. Dada la secuencia de observaciones y un modelo oculto de Markov $\lambda = (A, B)$, el algoritmo retorna la ruta a través del modelo oculto de Markov que asigne la máxima verosimilitud a la secuencia de observaciones. Recuperado de <https://machinelearningblogs.com/2016/12/01/opinion-mining-beyond-sentiment-analysis/>

En la cuenta <https://github.com/AustinRochford> se puede encontrar el algoritmo de Viterbi implementado en Python, utilizando como ejemplo de entradas datos de Wikipedia.

Este algoritmo también es conocido como el decodificador de Viterbi. Es muy utilizado en las telecomunicaciones para la decodificación de códigos convolucionales usados en las redes de telefonía celular **GSM** y **CDMA**. Es muy utilizado también para el reconocimiento del habla, lingüística computacional, bioinformática, entre otras.

10. Aplicaciones

10.1. Juegos

Los juegos generalmente presentan gran similitud con la realidad, en algunos casos simulan problemas de la vida real. Buenos juegos, están bien formulados, unen complejidad y simplicidad. Para algunos de los juegos no se conoce o no se tiene un algoritmo que garantice una solución (película de Netflix llamada *Black Mirror*). Jugar bien requiere inteligencia, permite de hecho desarrollar actividades cognitivas interesantes en el jugador. Son ejemplos claros de problemas de toma de decisiones. Muchos de ellos han servido como campo de estudio y de aplicación de técnicas de IA. Aquellos agentes diseñados para jugar juegos complejos (ajedrez) que sean capaces de mejorar su juego han servido de precursores para ser utilizados en dispositivos inteligentes.

Entre las características generales para el empleo de juegos en IA:

1. Las jugadas deben ser secuenciales y el resultado o efecto de una acción deben ser inmediato.
2. Son discretos, tiene un número finito de jugadas en cada estado del juego.
3. Son implícitos, no se conoce con anterioridad el valor de la ganancia obtenida por una jugada
4. Existe un conjunto de estados objetivos y la idea es conseguirlos, es decir que el objetivo general del juego es ganar, que el agente o el jugador consiga sus objetivos.

5. Son determinísticos, los resultados de las acciones son estados bien definidos.
6. Son accesibles, permiten percibir el estado actual y saber si se trata de un estado objetivo.

Dependiendo del número de jugadores, los juegos se caracterizan por ser unipersonales (un solo jugador) o para varios jugadores, bien sea en un mismo espacio físico o varios jugadores conectados en red. De la misma manera, dependiendo de la definición del juego, se pueden tener juegos con elementos al azar (por ejemplo, juego de dados) o sin elementos al azar (damas, ajedrez). El juego puede tener un entorno totalmente observable (ajedrez) o parcialmente observable (un laberinto, póker, etc.). Pueden tener además características de estático (laberinto, rompecabezas de 8 piezas), semi-dinámico con pequeños cambios previsibles (ajedrez), o dinámico, como es el caso de los video juegos.

10.2. Visión por computadora o visión artificial

Visión por computadora (*Computer Vision*) es un tópico multidisciplinario, que, basado en la formulaciones matemáticas y simulaciones físicas, tiene como objetivo analizar, extraer características y procesar imágenes o videos con el fin de interpretarlas e identificar los objetos inmersos en ellos tal como puede hacerlo un humano, de manera rápida y acertada. Shapiro and Stockman (2001) en su libro describen que el objetivo de la visión por computadora es el de permitir tomar decisiones útiles sobre los objetos físicos reales y escenas basadas en imágenes. Lo que se traduce en interpretar los objetos en las imágenes.

Un humano puede mirar una imagen y rápidamente identificar objetos, personas y describir la escena donde fue tomada la fotografía de una manera precisa. La visión por computadora pretende poder llegar a interpretar las imágenes de la misma forma como un humano puede llegar a interpretarlas. Gracias a los esfuerzos y el avance tecnológico tanto a nivel de *software* como de *hardware*, hoy en día es posible identificar computacionalmente algunos objetos, rostros, extraer modelos 3D de las escenas a partir de múltiples imágenes, caracterizar las imágenes, identificar los objetos y etiquetarlos de manera automática, analizar estados anímicos o emociones analizando los rostros e identificar especies de plantas, animales, entre otras cosas. Aun así, seguimos estando lejos o quizás cerca de lograr interpretar las imágenes como lo haría un humano, de forma general y en todos los escenarios posibles. Dichas técnicas han permitido hoy en día crear los automóviles autónomos e incluso humanoides, capaces de interpretar computacionalmente las imágenes con un éxito impresionante.

Muchos de los dispositivos que se usan hoy en día incluyen cámaras (teléfonos celulares, automóviles, etc.), de forma que el análisis y la interpretación de dichas imágenes se ha convertido en un área de estudio importante de la inteligencia artificial. En esta sección vamos a estudiar algunos métodos básicos de visión por computadora que utilizan técnicas de IA. Por ejemplo: para clasificar imágenes y extracción de características entre otras técnicas. Así como también conocer las técnicas para realizar tareas orientadas a la reconstrucción 3D.

Comencemos entonces a describir desde lo más básico, describiendo lo que es una cámara. Hoy en día una cámara es un dispositivo electrónico (*hardware*) que permite a través del uso de un lente

adquirir imágenes o videos (teléfonos celulares, drones, etc.). Los más recientes teléfonos celulares incluyen una cámara, algunos con dos lentes y resoluciones distintas. El avance tecnológico ha permitido desarrollar cámaras para celulares con muy buena resolución con la cual se pueden adquirir imágenes y videos con muy buena calidad.

Existen también otros dispositivos con los cuales se pueden adquirir distintas imágenes, por ejemplo, las usadas en el área de la medicina (tomógrafos, resonadores magnéticos, equipos de rayos X, ultrasonido, etc.) con los que se pueden visualizar los órganos del cuerpo humano, que después de ser procesados utilizando técnicas de procesamiento de imágenes y/o visión por computadora pueden ayudar al especialista a interpretarlas y dar un diagnóstico más acertado.

10.2.1. Formación de las imágenes

Las imágenes se forman como resultado de la proyección de una imagen sobre una película o proyector. Los elementos físicos que intervienen en la formación de la imagen son el objeto, la iluminación y el proyector o visualizador (donde se proyecta la imagen del objeto). La imagen se forma usando dispositivos de adquisición de imágenes, donde intervienen la iluminación, un lente, la distancia al objeto y la distancia focal (distancia del lente al objeto proyectado). Esto si estamos hablando de imágenes y videos adquiridos a partir de cámaras fotográficas. El proceso de adquisición de imágenes médicas es otro proceso de formación de las imágenes completamente diferente, en la que utilizando sensores de ultrasonido, tecnología magnética y rayos X (entre otros), se adquieren imágenes que muestran órganos o texturas que no podemos ver a simple vista como humanos. En esta guía solo hablaremos de las imágenes digitales adquiridas por cámaras fotográficas.

El principio de adquisición de las imágenes es fácilmente explicable y reproducible experimentalmente usando el denominado ejemplo de cámara *pinhole*. Consiste en una caja cerrada con el interior de la caja totalmente negro a la que se ha realizado un pequeño orificio. Por el orificio pasa la cantidad de la luz que emite el objeto, y en la cara final de la caja se proyecta la imagen (ver Figura 46).

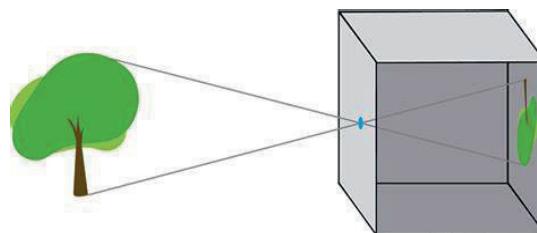


Figura 46. Modelo de cámara pinhole. Caja negra con un orificio pequeño por donde entra la luz y se proyecta la imagen al fondo de la caja. El fondo de la caja puede contener una película fotosensible y la imagen se proyectará allí, de forma inversa, luego puede ser revelada en un cuarto oscuro y obtener la imagen capturada. Recuperado de <https://unifeed.club/view/842724-camara-oscura-como-funciona/>

El modelo de cámara *pinhole* es el modelo utilizado inicialmente por las cámaras fotográficas. Detrás de este modelo de cámara *pinhole* existe una matemática básica (ver Figura 47).

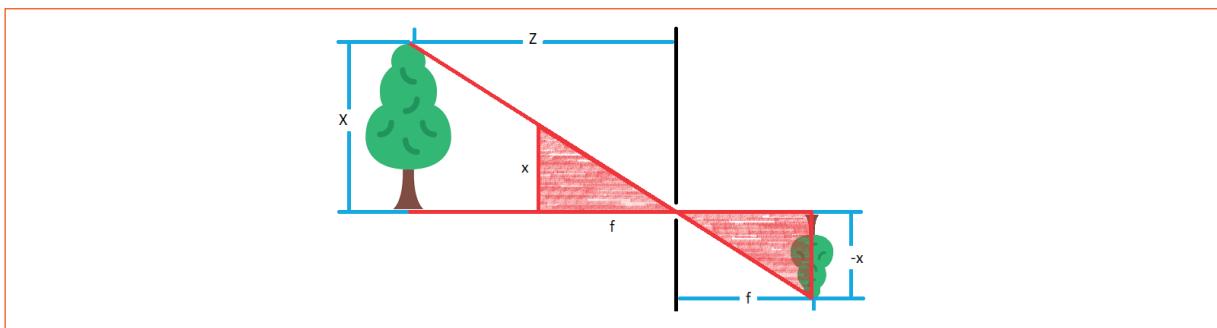


Figura 47. Formulación matemática de la cámara pinhole.

Donde X es la altura del objeto real, $-x$ la altura del objeto proyectado, Z la distancia del objeto al centro de proyección (agujero en la caja) y f la distancia entre el centro de proyección y el objeto proyectado (distancia focal). El sistema de ecuaciones que relaciona estas cuatro variables viene determinado por:

$$\frac{X}{Z} = \frac{-x}{f}$$

Despejando x queda la siguiente ecuación:

$$x = \frac{f}{Z} X$$

A mayor distancia del objeto a la cámara la imagen proyectada será más pequeña, a menor distancia focal, la imagen proyectada será más grande. Por supuesto el tamaño del objeto también influye en el tamaño del objeto proyectado. Ahora podemos describir lo que es la proyección en perspectiva.

10.2.2. Proyección en perspectiva

La proyección en perspectiva nos dice que en cualquier cámara incluyendo el modelo de cámara pinhole, el tamaño del objeto proyectado de cualquier objeto varía según la distancia de la cámara al objeto, donde el tamaño del objeto proyectado es proporcional al objeto real, e inversamente proporcional a la distancia de la cámara al objeto. Entonces si alejamos un objeto de la cámara lucirá más pequeño y si lo acercamos a la cámara se verá más grande.

Las imágenes tienen dos dimensiones, la ley de perspectiva de proyección se aplican para cada espacio en x y en y . Tenemos entonces que:

$$x = \frac{f}{Z} X, \quad y = \frac{f}{Z} Y$$

En ambos casos el tamaño del objeto proyectado varía según la distancia del objeto a la cámara y la distancia focal. Una consecuencia de la ley de proyección en perspectiva es que las líneas paralelas en una dirección convergen en el infinito, este punto de convergencia es denominado punto de fuga.

10.2.3. Punto de fuga

El punto de fuga es el punto en donde convergen las líneas paralelas a una dirección dada en el espacio y que no son paralelas al plano de proyección. En cada dirección en el espacio 3D por ejemplo pueden definirse hasta tres puntos de fuga. Líneas paralelas en el mundo real, a la distancia convergen geométricamente en un punto (ver Figura 48).

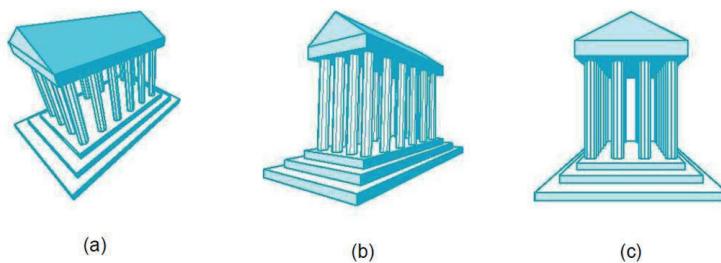


Figura 48. Proyección con tres puntos de fuga (a), dos puntos de fuga (b) y un punto de fuga (c). Recuperado de Angel and Shreiner (2012).

Hasta ahora hemos hablado de la imagen proyectada usando el modelo de cámara *pinhole* que tiene una limitación fundamental y es que muy pocos rayos de luz permiten proyectar la imagen. Una cámara fotográfica que se fundamenta básicamente en el modelo de cámara de *pinhole*, utiliza una lente, esta lente le permite transmitir por refracción de la luz, los rayos de luz a la cámara, enfocarlos y formar la imagen. A mayor distancia del objeto a la cámara el desenfoque del objeto proyectado será mayor, debido a que los rayos de luz no coinciden en un mismo punto según varía la distancia del lente al objeto. Las cámaras digitales permiten enfocar la imagen ajustando el lente para que a una distancia dada del objeto a la cámara aún se pueda mantener la imagen enfocada (ver Figura 49).

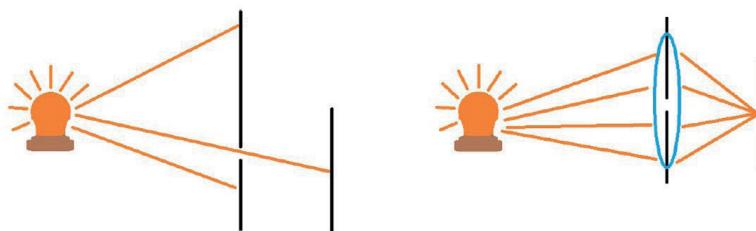


Figura 49. Enfoque y uso del lente en las cámaras fotográficas.

10.2.4. Imagen digital

Tal como lo describen González & Woods (2002), una imagen digital puede ser definida por una función bidimensional, generalmente representada como: $f(x,y)$ or $I(x,y)$. Donde x e y son las coordenadas del plano de la imagen y la función representa el valor de la intensidad de la imagen de niveles de gris o de color en dicho punto. En el mundo real hay infinitas variaciones de intensidad tanto de colores como en niveles de gris, sin embargo, como humanos tenemos un límite de percepción de intensidades.

Así mismo para representar computacionalmente dichos valores de intensidad, los valores de f, x e y , deben ser finitos y valores discretos. Una imagen digital es representada por una matriz de valores discretos, donde x e y describen la dimensión de la imagen (tamaño) (ver Figura 50). Es así como una imagen digital está compuesta por un número finito de elementos. Cada elemento pertenece a una posición x e y en la imagen. Estos elementos son llamados píxeles. Cada valor $f(x,y)$ corresponde al valor de un píxel. Un píxel es la representación mínima de una imagen digital, cuyo valor representa el valor de intensidad en el plano de la imagen en la posición (x,y) .

$$f(x,y) = \text{valor de intensidad (pixel) en la posición } (x,y) \text{ en el plano de la imagen}$$

La representación de un píxel depende del número de bits usados para representarlo computacionalmente: 8-bit, 16-bit, 12-bit, etc. Este valor depende del dispositivo utilizado para adquirir la imagen.

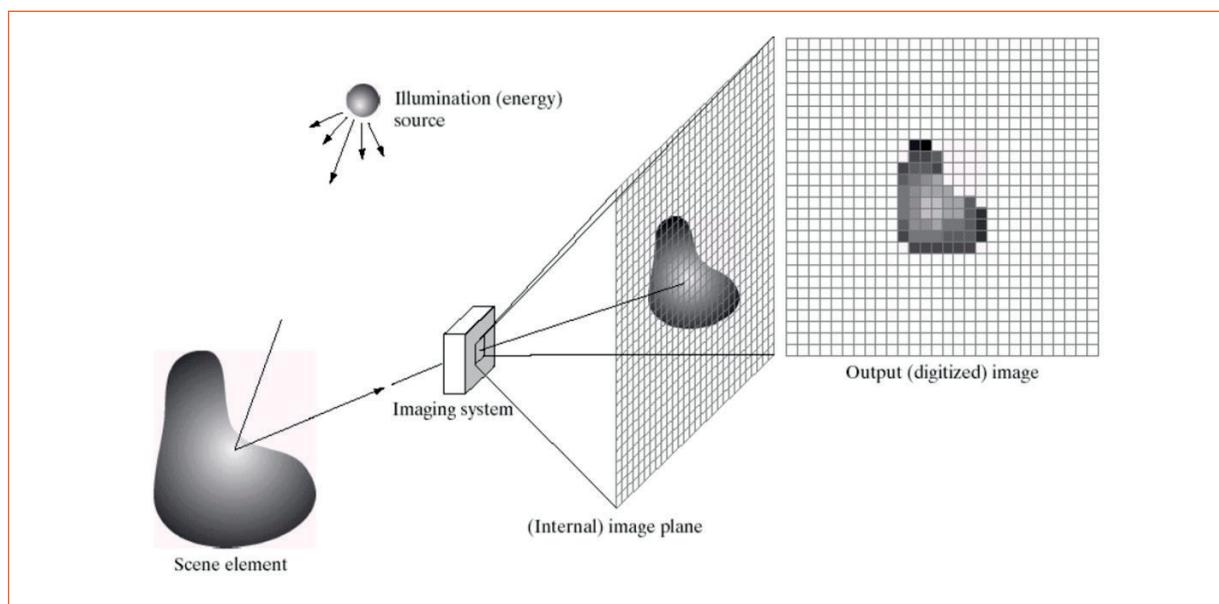


Figura 50. Imagen digital generada por la proyección de una escena 3D al plano de proyección 2D. Recuperado de González & Woods (2002).

En términos generales existen tres tipos de imágenes:

- Las **imágenes binarias** son aquellas cuyos valores de los píxeles solo es representado por negro o blanco. Dependiendo de la representación de un píxel en el computador el valor mínimo (0) es el valor de intensidad negro y el máximo valor (1 o 255) es el color blanco (ver Tabla 8).
- Las **imágenes en niveles de gris** son representadas por el rango 0 al máximo valor de representación computacional (255, 65535, etc., tal como se describe en la Tabla 8).
- Las **imágenes a color** son representadas por al menos tres canales, cada canal representa un componente de color RGB (*red, green* y *blue*); y cada componente de color varía entre el mínimo valor (0) y el máximo valor de representación.

Un segundo propósito es la de reconstruir objetos en 3D; si se toman varias imágenes de un objeto a partir de diferentes perspectivas, o utilizando múltiples cámaras y reconstruir tridimensionalmente objetos a partir de dichas proyecciones 2D de imágenes. También podríamos querer hacer análisis en movimiento para estudiar el movimiento de ciertos objetos. Este es un problema común en visión por computadora, donde se analiza una secuencia de imágenes a partir de un video y los objetos tienen cierto movimiento.

Tabla 8
Representación computacional de un valor de intensidad de la imagen

		Negativos	Positivos
		Negativos reales (VN)	Positivos Falsos (FP)
Negativo	Negativo	Negativos reales (VN)	Positivos Falsos (FP)
	Positivo	Falsos Negativos (FN)	Positivos reales (VP)

La visión por computadora tiene varios retos importantes a tomar en cuenta, algunos autores la llaman **invariaciones**. Las invariaciones en los objetos; son aquellas variaciones del objeto, bien sea geométricas, de iluminación, de formas que aun con dichas variaciones es posible reconocerlo. Desarrollar un algoritmo que sin importar lo variante que se puede ver el objeto en la imagen y aun así poder reconocer el objeto es uno de los mayores desafíos en visión por computadora. Algunas de esas invariaciones son:

a. Escala

Objetos vistos en diferentes escalas y aun así reconocer que se trata de una bicicleta (ver Figura 51).



Figura 51. Objeto visto en la imagen en diferentes escalas. Recuperado de <https://www.educima.com/dibujo-para-colorear-bicicleta-i16111.html>

b. Rotación

Así como el factor escala, el factor de rotación de la imagen permite ver la imagen con ángulos de rotación distintos. El reto es reconocer que se trata del mismo objeto (ver Figura 52).



Figura 52. Objeto visto en la imagen visto en diferentes orientaciones. Recuperado de <https://www.educima.com/dibujo-para-colorear-bicicleta-i16111.html>

c. Iluminación

Imagen con diferentes efectos de iluminación conteniendo los mismos objetos (ver Figura 53).

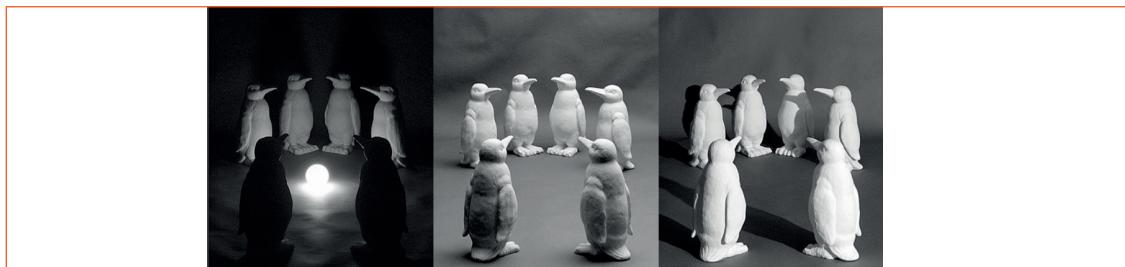


Figura 53 Diferentes efectos de iluminación. Recuperado de http://vision.stanford.edu/teaching/cs231a_autumn1112/lecture/lecture1_introduction_cs231a.pdf

d. Deformación

Objetos vistos en diferentes estados de formación de su apariencia (ver Figura 54).



Figura 54. Pintura de Xu, Beihong 1943. Recuperado de http://vision.stanford.edu/teaching/cs231a_autumn1112/lecture/lecture1_introduction_cs231a.pdf

e. Oclusión parcial

Imagen con algunos objetos ocluyendo el objetivo principal de la imagen (mujer a caballo, Figura 55).

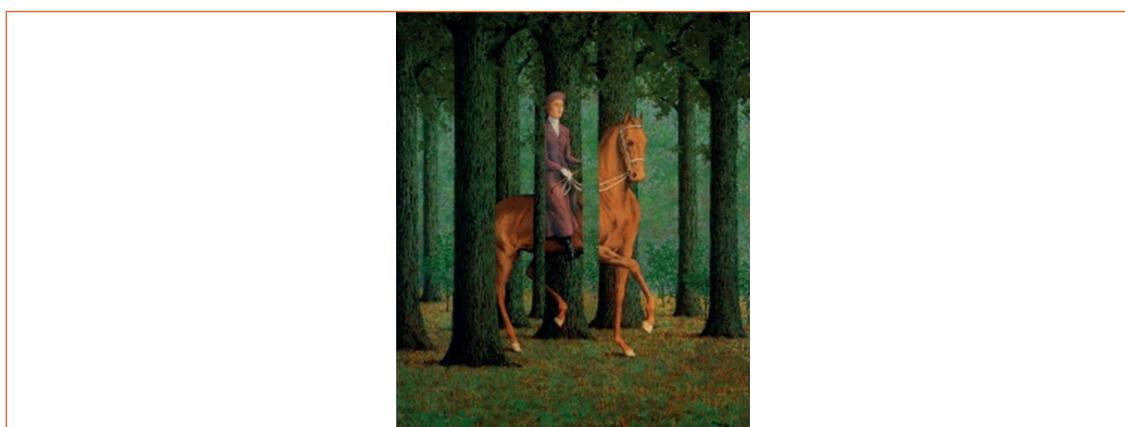


Figura 55. The Blank Signature (La Carte Blanche) 1965 por Rene Magritte. Recuperado de http://vision.stanford.edu/teaching/cs231a_autumn1112/lecture/lecture1_introduction_cs231a.pdf

f. Puntos de vista

Vista del mismo objeto desde distintos puntos de vista (ver Figura 56).

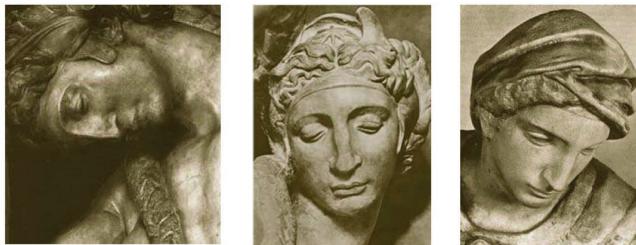


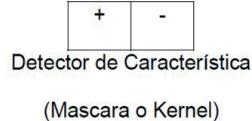
Figura 56. Michelangelo 1475-1564. Recuperado de http://vision.stanford.edu/teaching/cs231a_autumn1112/lecture/lecture1_introduction_cs231a.pdf

10.2.5. Extracción de características

Una de las tareas más comunes que realiza la visión por computadora es la de caracterizar las imágenes, extrayendo o indicando características propias de la imagen que permitirán tomar decisiones locales sobre si un punto en la imagen cumple o no con algunas características. Esto permitirá realizar clasificaciones de imágenes, identificación de objetos, segmentar semánticamente las imágenes, entre otras cosas. Por ejemplo, identificar los bordes en la imagen, esquinas, regiones, crestas, etc.

255	212	7	1	3
211	237	3	9	0
218	240	8	12	2
240	241	8	4	0

Imagen A



43	205	-1	-2	
-26	234	-6	9	
-22	232	-4	10	
-1	233	-4	4	

Imagen Resultante

Figura 57. Resultado de aplicar un detector de característica sobre los datos de la imagen A.

Claramente en este ejemplo de imagen (ver Figura 57) hay una región que es más brillante que la otra región y queremos caracterizar los bordes. Usamos un detector de característica correspondiente que permitirá calcular la diferencia de valores de píxeles vecinos de manera vertical. Claramente, la columna 2 de la imagen resultante caracteriza la imagen original diciendo que claramente allí hay un borde entre las columnas 2 y 3 de la imagen original. Generalmente, se toma el valor absoluto de la diferencia de tal manera que los valores de densidad quedarán con valores positivos. Estos operadores lo que buscan es resaltar dónde existían diferencias pronunciadas de valores de densidad, indicando allí bordes en la imagen (ver Figura 58).

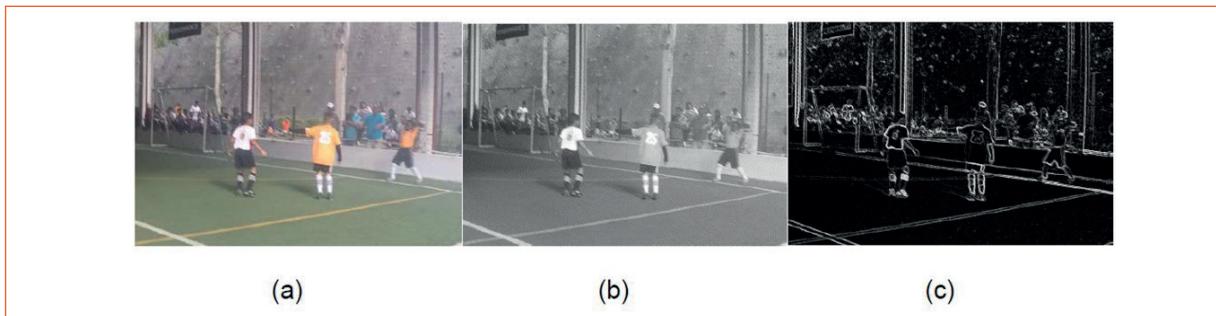


Figura 58. Imagen original en color (a). Imagen convertida a escalas de grises (b). Imagen con bordes detectados (c).

Generalmente, el gradiente de la imagen se calcula tanto en la dirección de x como en la dirección de y. Calculando los bordes en cada dirección. Si queremos calcular el gradiente en ambas direcciones entonces debemos combinar la aplicación del gradiente tanto en x como en y, aplicando la siguiente formulación:

$$Gx = I(x,y) \otimes [-1 \ 1],$$

$$Gy = I(x,y) \otimes \begin{bmatrix} +1 \\ -1 \end{bmatrix},$$

$$|G| = |Gx| + |Gy|$$

Lo que se traduce a:

$$Gx = I(x,y) \otimes Kx,$$

$$Gy = I(x,y) \otimes Ky$$

Donde Kx y Ky son el kernel en x e y respectivamente. Es así como la misma formulación puede ser aplicada a distintas configuraciones de *kernel*. Algo importante aquí es el operador utilizado para caracterizar la imagen. Este operador es llamado *kernel*. Según cómo se defina el *kernel* la imagen resultante será distinta. El operador utilizado en la Figura 58 es el operador de Sobel:

Gx	<table border="1" style="border-collapse: collapse; width: 100%; height: 100%;"> <tr><td>-1</td><td>0</td><td>+1</td></tr> <tr><td>-2</td><td>0</td><td>+2</td></tr> <tr><td>-1</td><td>0</td><td>+1</td></tr> </table>	-1	0	+1	-2	0	+2	-1	0	+1	Gy
-1	0	+1									
-2	0	+2									
-1	0	+1									
	<table border="1" style="border-collapse: collapse; width: 100%; height: 100%;"> <tr><td>+1</td><td>0</td><td>-1</td></tr> <tr><td>+2</td><td>0</td><td>-2</td></tr> <tr><td>+1</td><td>0</td><td>-1</td></tr> </table>	+1	0	-1	+2	0	-2	+1	0	-1	
+1	0	-1									
+2	0	-2									
+1	0	-1									

El operador gradiente combina el cálculo del gradiente en x y en y, en ambas direcciones del gradiente, vertical y horizontal, a fin de obtener una mejor detección de borde tanto horizontal como vertical. Algunas otras configuraciones de *kernel* son:

Prewitt:

Gx	<table border="1" style="border-collapse: collapse; width: 100%; height: 100%;"> <tr><td>+1</td><td>0</td><td>-1</td></tr> <tr><td>+1</td><td>0</td><td>-1</td></tr> <tr><td>+1</td><td>0</td><td>-1</td></tr> </table>	+1	0	-1	+1	0	-1	+1	0	-1	Gy
+1	0	-1									
+1	0	-1									
+1	0	-1									
	<table border="1" style="border-collapse: collapse; width: 100%; height: 100%;"> <tr><td>+1</td><td>+1</td><td>+1</td></tr> <tr><td>0</td><td>0</td><td>0</td></tr> <tr><td>-1</td><td>-1</td><td>-1</td></tr> </table>	+1	+1	+1	0	0	0	-1	-1	-1	
+1	+1	+1									
0	0	0									
-1	-1	-1									

Kirsh:

$$\begin{array}{c}
 G_x \\
 \begin{array}{|c|c|c|} \hline +5 & -3 & -3 \\ \hline +5 & 0 & -3 \\ \hline +5 & -3 & -3 \\ \hline \end{array}
 \end{array}
 \quad
 \begin{array}{c}
 G_y \\
 \begin{array}{|c|c|c|} \hline +5 & +5 & +5 \\ \hline -3 & 0 & -3 \\ \hline -3 & -3 & -3 \\ \hline \end{array}
 \end{array}$$

10.2.6. Filtros lineales

Teóricamente, los filtros aplicados a las imágenes son modelados como sistemas lineales y no lineales (Szeliski (2010)). Los filtros lineales sobre las imágenes generalmente son usados para añadir ruido, suavizar los bordes de la imagen, resaltar los bordes y esquinas o eliminar ruido. Básicamente los filtros generan una imagen cuyos valores de píxeles son la combinación lineal o no lineal de un píxel con los píxeles de su entorno. Los operadores lineales estiman la sumatoria resultante de operar los píxeles vecinos y los operadores no lineales toman el mínimo o el máximo valor resultante de sus píxeles vecinos. La vecindad a tomar en cuenta es determinada por el tamaño del *kernel* u operador. Matemáticamente la ecuación para un operador de filtro lineal viene dada por la ecuación:

$$g(x,y) = \sum_{k,l} f(i+k, j+l) h(k, l)$$

Donde, el operador, también llamado *kernel* o máscara, viene dado por $h(k, l)$. Normalmente, el tamaño del *kernel* es una matriz 3x3, 5x5, 7x7, etc. El *kernel* también es considerado como los coeficientes del filtro. A la ecuación antes descrita se la denomina **operación de convolución** (ver Figura 59).

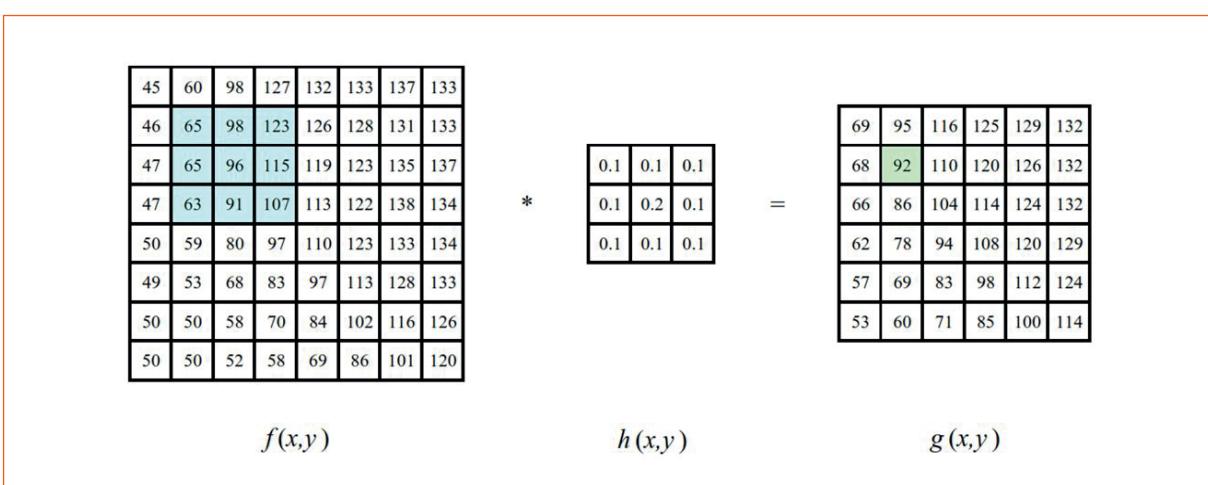


Figura 59. Operador de filtro lineal u operador de convolución. Recuperado de Szeliski (2010).

De esta manera, construimos los operadores de los filtros, definiendo un *kernel* y ejecutando el operador convolución. Generalmente h es la función de respuesta al impulso, esto es debido a que

la function h convoluciona con la señal del impulse $\delta(i,j)$, la cual es una imagen con píxeles cuya vecindad tienen el valor 0, y el píxel en el medio el valor 1. Tal como se indica en la matriz:

0	0	0
0	1	0
0	0	0

$\delta(i,j)$

El principal problema son los extremos de las imágenes, dado que los bordes de los píxeles extremos de la imagen no tienen su vecindad completa, que cubra el *kernel*, resultando en una imagen mucho más pequeña que la original. Para compensar este problema hay varias maneras de llenar los píxeles faltantes:

- Asignando un valor constante en los extremos; por ejemplo, asignarle 0 o 255.
- Reemplazar el valor del píxel tratando los bordes como una extensión de su píxel más cercano.
- No se reemplaza se extiende la imagen tomando el mismo valor del píxel de la imagen original.

10.2.7. Filtro gausiano

El filtro gausiano es un filtro lineal que aplicado a la imagen suaviza los bordes. Este filtro permite modelar la distribución gausiana sobre la imagen y es probablemente el filtro más usado. Generalmente se usa cuando queremos:

- a. Remuestrear la imagen a una menor resolución, y evitar cualquier problema de *aliasing* al reducir el muestreo de la imagen.
- b. Reducir el ruido en la imagen.

El grado de suavidad del filtro gaussiano viene dado por la desviación estándar de la función de Gauss, por otro lado, a mayor valor de la desviación estándar mayor tamaño tendrá el *kernel*. El filtro gaussiano promedia los píxeles vecinos y realza el valor del píxel central al *kernel*. De esta manera preserva los bordes, mientras suaviza su vecindad.

10.2.8. Detector de bordes de Canny

Hoy en día se han definido buenos detectores de bordes, mejores que los detectores de bordes horizontal y vertical como lo es el filtro de gradiente. Uno de ellos es el método de detección de bordes de Canny (1986). Este método ha resultado ser más efectivo que algunos otros detectores de bordes, porque además de utilizar la magnitud del gradiente, traza áreas para encontrar máximos locales y trata de conectarlos de tal manera que mantiene la continuidad de los bordes. Cuando varios bordes se encuentran en un punto, no es tomado en cuenta como borde, pero cuando los bordes son simples bordes, son bien definidos (ver Figura 60).



(a)



(b)

Figura 60. (b) Resultado de aplicar el método de detección de bordes de Canny a la imagen (a).

10.2.9. Detector de esquinas de Harris

Otra característica importante en visión artificial es la detección de esquinas. El método de Harris para detección de esquinas permite identificar los puntos donde se cruzan dos bordes para formar una posible esquina. Este método puede ser aplicado para imágenes del tipo cuadrillas como el modelo de la tabla de ajedrez. Algunas veces es necesario conseguir puntos correspondientes en distintas imágenes, en secuencias de imágenes en un mismo ambiente, a fin de conocer cómo se relacionan o qué tanto se relacionan dos imágenes. Dichos puntos correspondientes en imágenes se les denomina puntos característicos y deben ser puntos únicos reconocibles. El método de Harris hace un rastreo sobre la imagen utilizando una ventana, desplaza dicha ventana en una dirección y calcula la variación de intensidad en dicha ventana (ver Figura 61), sabiendo que las esquinas representan una variación importante en el gradiente de la imagen. Para ello calcula los autovalores de los gradientes en cada píxel, de esta manera toma en cuenta orientaciones distintas a las direcciones de las coordenadas en x y en y . Por cada píxel se obtienen dos autovalores. Si ambos autovalores son máximos locales entonces dicho punto es candidato a ser una esquina.

$$\text{Autovalores} \rightarrow \begin{bmatrix} \sum G_x^2 & \sum G_x G_y \\ \sum G_x G_y & \sum G_y^2 \end{bmatrix}$$

Donde G_x y G_y , son el gradiente en los puntos x e y de la imagen. La sumatoria indica la sumatoria de los valores de gradiente en una ventana determinada.



Figura 61. Resultados de aplicar el método Harris. (a) y (c) fotos originales y (b) y (d) después de aplicar el método de Harris. Recuperado de <https://www.houseofstaunton.com/masters-chess-table.html>

10.2.10. Detector de características actuales

Los métodos de detección de características son básicamente extensiones del método de detección de esquinas de Harris. Normalmente son:

- a. Localizables.
- b. Identificador único (resumen la identidad de la característica, típicamente invariantes a la iluminación, orientación, escala o traslación).
- c. Los métodos más comunes que la gente generalmente usa son:
 - HoG (Histograma de Gradientes orientados – *Histogram of Oriented Gradients*)
 - SIFT (Transformada de escala invariante – *Scale Invariante Feature Transform*)

Estos métodos reducen la influencia de las variaciones como la rotacional, calculando estadísticas que son invariantes a efectos como la rotación, escala, y ciertas transformaciones de perspectivas. Aquí un ejemplo ilustrado por Ian London y publicado en su cuenta github (ver Figura 62). El problema consiste en encontrar los puntos correspondientes entre las dos imágenes con diferentes perspectivas.

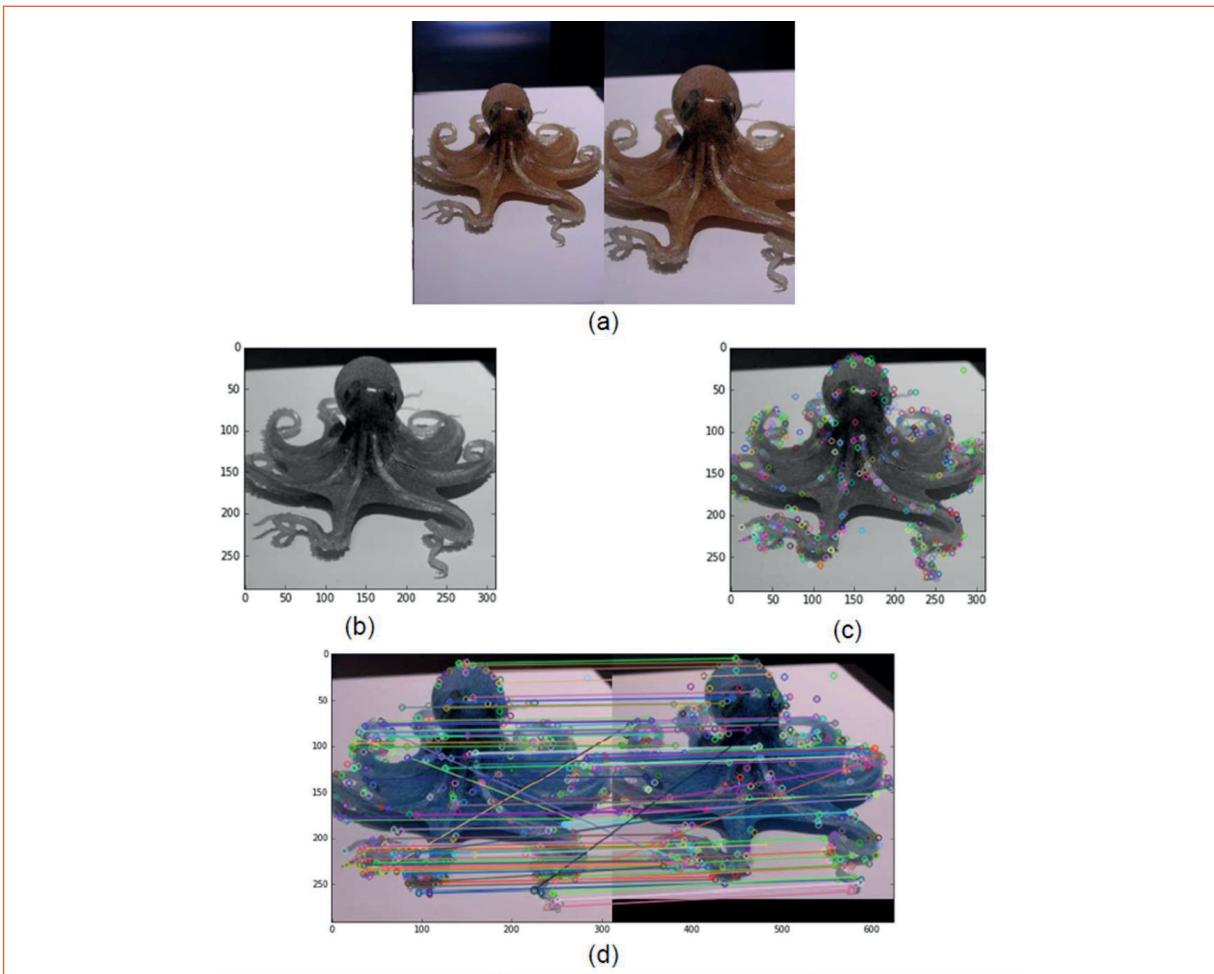


Figura 62. Resultados de aplicar el SIFT implementado en Python. Recuperado de <https://ianlondon.github.io/blog/how-to-sift-opencv/>

10.3. Robótica

La robótica data de hace bastante tiempo. Actualmente se han construido desde los operadores de trenes y autos autónomos, hasta una robot con ciudadanía (Sophia). Sophia es una robot creada en base a una combinación de técnicas de IA, como redes neurales, sistemas expertos, máquina de percepción, procesamiento de lenguaje natural, control de monitor adaptativo y arquitectura cognitiva entre otras técnicas. Recientemente le otorgaron la ciudadanía en Egipto (ver Figura 63).



Figura 63. Robot Sophia creado por el grupo Hanson Robotic. Fuente: <https://www.hansonrobotics.com/sophia/>

Cuando se habla de robótica, nos viene a la mente generalmente, la imagen del robot de la famosa película “La guerra de las galaxias”, un robot que parecía tener hasta sentimientos. Aun así, la robótica sigue un continuo avance y evolución.

La robótica es la ciencia del campo de la tecnología que estudia desde el diseño y la construcción de máquinas capaces de realizar tareas que el humano hace regularmente. Se basa fundamentalmente en el álgebra, los autómatas programables, la máquina de estados, la mecánica, la electrónica y la informática. **Básicamente la robótica es un conjunto de conocimientos teóricos y prácticos que permite el desarrollo de sistemas basados en estructuras mecánicas, articulares, con un cierto dote de inteligencia y autonomía para realizar tareas a nivel industrial, macro, micro y hoy en día a nivel de nanotecnología, tareas que en algunos casos el hombre puede realizar regularmente y otras en las que para el hombre es difícil de realizar.** Un robot es capaz de recibir información a través generalmente de los sensores, procesar la información, tomar una decisión y luego ir a la acción. Requiere entonces de la planificación, percibir información del entorno, del estado actual de su entorno, para tomar una decisión.

En el Diccionario inglés Oxford, se acuña a Isaac Asimov como el creador de la palabra robótica (Macchiavello, 2008). Asimov fue quien planteó algunos axiomas básicos para el funcionamiento de un robot:

1. Un robot nunca debe hacer daño a un ser humano o permitir que se le haga daño por no actuar.
2. Un robot debe obedecer las órdenes dadas por un ser humano, siempre y cuando no contradiga el primer axioma.
3. Un robot debe proteger su existencia en la medida en que no contradiga los primeros dos axiomas.

Sin embargo, estas leyes suponen un robot inteligente y en muchos casos no suele ser tan fácil de crear un robot que cumpla con todos estos axiomas. Como es en los casos en que el salvar a un

humano incluya el destruir al otro, bien sea por el valor del humano al que intenta salvar o por la evaluación que pueda hacer el robot, en cuanto a lo que para él significa humanidad.

La robótica hoy en día es una ciencia que ha tenido grandes avances, sin embargo, aún ofrece un campo amplio para el desarrollo e innovación tecnológica. Esto es una motivación fuerte para los investigadores para querer seguir desarrollando robots más evolucionados y complejos.

A pesar de que las películas de ciencia ficción han hecho ver a los robots como malvados por naturaleza, los robots solo son máquinas que hacen lo que están programadas a hacer. Es el humano quien programa a los robots. En algunos casos como el de Sophia, un robot que aprende y que responde según su conocimiento y aprendizaje obtenido. Sin embargo, hay cosas básicas y leyes que en teoría debe cumplir en función de lo programado.

Existen las máquinas autómatas programables, que son máquinas electrónicas diseñadas para controlar un proceso en tiempo real, carecen de inteligencia, actúan y reaccionan igual ante sucesos iguales. Generalmente un autómata forma parte de un robot, dedicándose a controlar señales y actuar en función de la respuesta a las señales percibidas, por ejemplo, permiten la manipulación de bazos mecánicos.

10.3.1. Clasificación de los robots

En la actualidad no existe una estandarización de clasificación de los robots, pero entre los más comunes se tienen:

- **Robots Manipuladores:** poseen un sistema de control sencillo, casi manual de secuencia fija o variable.
- **Robot de Aprendizaje:** por medio de la repetición de movimientos y acciones ejecutada previamente por un humano, el robot sigue y memoriza los movimientos.
- **Robots con control sensorizado:** por medio de un controlador que recibe órdenes, las envía a un manipulador para que realice el movimiento o acción.
- **Robots inteligentes:** además de poseer las características de los robots descritos anteriormente, poseen sensores que procesan, toman decisiones y actúan en tiempo real.

10.3.2. Clasificación de los robots según su arquitectura

La arquitectura de los robots es definida por la configuración general del mismo, por su forma y elementos o atributos que posea:

- **Poliarticulados:** su característica principal es que son sedentarios, eventualmente pueden ser guiados para realizar desplazamientos limitados (robots industriales, cartesianos, etc.).
- **Móviles:** poseen gran capacidad de desplazamiento, utilizan un sistema locomotor tipo rodante, se mueven mediante la información que perciben de su entorno a través de sensores. Es una arquitectura de las precursoras en el desarrollo de la IA en la Universidad de Stanford.

- **Androïdes:** este tipo de robot intenta reproducir parcial o totalmente el comportamiento cinemático del ser humano. El más evolucionado hoy en día es justamente Sophia.
- **Zoomórficos:** son los tipos de robots que imitan el movimiento o desplazamiento de algunos seres vivos, perro, arañas, etc.
- **Híbridos:** son aquellos de difícil clasificación pues pueden contener una variación de los otros robots o con características arquitectónicas diferentes.

Otras características de clasificación de los robots pueden ser:

1. Propósito o función
2. Sistema de coordenada empleado
3. Número de grados de libertad
4. Sistema de control

Glosario

AlexNet

Es el nombre de una red CNN ya entrenada utilizada para el reconocimiento de objetos en las imágenes.

CDMA

Siglas de *Code Division Multiple Access* (Acceso Múltiple por División de Código) es un método de acceso por código usado por los sistemas móviles de comunicación.

CNN

Es el acrónimo en inglés de *Convolutional Neural Networks*, redes neurales convolucionales. Es una red neural que se utiliza para aplicar técnicas de *deep learning*.

Constituye uno de los métodos para hacer reconocimiento de imágenes, clasificación de imágenes, detección de objetos, reconocimiento de rostros, etc.

Deep Learning

Disciplina de aprendizaje automático no supervisado capaces de aprender sin supervisión humana, permitiéndoles realizar clasificaciones según características comunes entre los objetos. Requiere para ello grandes volúmenes de datos y un buen hardware para llegar a una solución general.

GoogleLetNet

Es también el nombre de una red CNN entrenada utilizada para la clasificación de imágenes, diseñada por Google.

GSM

Global System for Mobile o sistema global para las comunicaciones móviles.

IA

Acrónimo de Inteligencia Artificial, en español. En inglés suele ser AI del inglés *Artificial Intelligence*.

Mathworks

El nombre proviene de The MathWorks, Inc. y es una corporación privada estadounidense que se especializa en *software* de computación matemática. Sus principales productos incluyen MATLAB y Simulink, que admiten análisis de datos y simulación. Contiene varios paquetes para la aplicación de procesamiento de imágenes, señales, redes neurales, aprendizaje automático, supervisado no supervisado y aprendizaje profundo y *deeplearning* entre otros.

NP

En teoría de la complejidad computacional es un área de la teoría de la computación que estudia la clasificación de la complejidad de los problemas computacionales. Los problemas computacionales se observan en las relaciones entre un conjunto de instancias y un conjunto de soluciones. Los problemas computacionales según su complejidad se clasifican en Clase P, NP, los problemas de complejidad NP son problemas de decisión no determinísticos.

NP-completos

Son un tipo más complejo de problemas no determinísticos computacionales que los problemas NP. Donde la diferencia de complejidad entre un problema y otro es de la forma polinomial.

Rutas Michelin

Son mapas viales realizados por la empresa Michelin, empresa de comercialización de neumáticos

Enlaces de interés

Artificial Intelligence: A Modern Approach

En esta página se publica todo lo referente al libro principal de referencia de este curso.

<http://aima.cs.berkeley.edu/>

Mathworks

Sitio oficial de Mathworks latinoamerica, desarrolladores de la aplicación MatLab, diseñada para el análisis de datos, señales e imágenes. Poderosa herramienta usada por investigadores en distintas áreas; entre esas el área de inteligencia.

<https://la.mathworks.com/>

PyImageSearch

Sitio web dedicado para mostrar de manera amena y sencilla el desarrollo de aplicaciones en el área de procesamiento de imágenes en las que se utiliza inteligencia artificial. Es un sitio dedicado a programadores. Es un blog personal alimentado por el Dr. Adrian Rosebrock.

<https://www.pyimagesearch.com/>

Robot Sophia

Sitio Web donde se describe quién es Sophia y qué técnicas de IA fueron utilizadas para su funcionamiento.

<https://www.hansonrobotics.com/sophia/>

Watson from IBM

La supercomputadora capaz de aprender a partir de la acumulación de información es capaz de interactuar con un humano usando lenguaje natural.

<https://www.ibm.com/watson>

Bibliografía

Referencias bibliográficas

Angel, E. and Shreiner, D. (2012). *Interactive Computer Graphics 6E*. London: Pearson.

Arthur, S (1959). *Some Studies in Machine Learning Using the Game of Checkers*. IBM Journal of Research and Development. 3(3): 210–229. CiteSeerX 10.1.1.368.2254. doi:10.1147/rd.33.0210.

Audi, R. (1995). *Ockham's razor*. The Cambridge Dictionary of Philosophy (en inglés) (2nd edition). Cambridge: Cambridge University Press.

Berzal Fernando (2017). *Breve historia de la Inteligencia Artificial: el camino hacia la empresa*. Recuperado de <http://asesoresdepymes.com/breve-historia-la-inteligencia-artificial-camino-hacia-la-empresa/>.

Canny, J. (1986). *A Computational Approach to Edge Detection*. IEEE Trans. Pattern Analysis and Machine Intelligence, 8(6):679–698.

Chomsky, N. (1956). *Three Models for the Description of Language*. IEEE Trans. on Info. Theory; 2:3.

Craik K. (1967). *The Nature of Explanation*. Cambrige: Cambridge University Press.

Forney, G. D. (1973). *The Viterbi algorithm*. Proceedings of the IEEE, 61(3), 268-278.

Gómez I. (2019). *Máster oficial en gestión y análisis de grandes volúmenes: Big Data*. Recuperado de https://info.escueladenegociosydireccion.com/master-big-data/?utm_source=RevistaENyD&utm_medium=referral&utm_content=M%C3%A1sterBigData18

Gonzalez and Woods (2018), *Digital Image Processing 4th Edition*. London: Pearson/Pretince Hall.

Haugeland, J. (Ed.). (1985). *Artificial Intelligence: The Very Idea*. Cambridge, Massachusstes: MIT Press.

IBM (2011). *Watson*. Extraída de: <https://www.ibm.com/watson>

Lederberg, Joshua (1987). *How Dendral Was Conceived and Born*. ACM Symposium on the History of Medical Informatics, 5 November 1987, Rockefeller University. New York: National Library of Medicine.

Levinson, S. E. (1986). *Continuously variable duration hidden Markov models for automatic speech recognition*. Computer Speech & Language, 1(1), 29-45.

Luger, G. F. (Ed.). (1995). *Computation and intelligence: Collected readings*. Palo Alto: AAAI Press.

Macchiavello, T. (2008). *Monografía Robótica*. Recuperado de: <https://www.monografias.com/trabajos31/robotica/roboticas.html>

McCulloch & Pitts (1943). *A logical calculus of the ideas immanent in nervous activity*. Bulletin of Mathematical Biophysics, 5:115-133.

Miller G. (1956). *The magical number seven, plus or minus two: Some limits on our capacity for processing information*. Psychological Review. 63:81–97.

Minsky, M. L. and Papert, S. (1969). *Perceptions: An Introduction to Computational Geometry (first edition)*. Cambridge, Massachusetts: MIT Press.

Newell, A. and Simon, H.A. (1957). *The Logic Theory Machine. A Complex Information Processing System*. Journal of Symbolic Logic 22 (3):331-332.

Rabiner L. R (1989). *A tutorial on Hidden Markov Models and Selected Applications in Speech Recognition*. Proceeding of IEEE. 77(2).

Russell S. and Norvig P (2010). *Artificial Intelligence. A Modern Approach, Third Edition*. New Jersey: Prentice Hall.

Shapiro, L. and Stockman, G. (2001). *Computer Vision*. New Jersey: Prentice Hall.

Shoup, Victor (2008). *Euclid's algorithm. A Computational Introduction to Number Theory and Algebra*. Cambridge: Cambridge University Press.

Shwab, K. (2016). *La Cuarta Revolución Industrial de Klaus Shwab*. México: Editorial Debate.

Stanford Racing Team (2006). *Stanley*. Recuperado de <https://cs.stanford.edu/group/roadrunner/stanley.html>

Steven K. (2001). *The Ultimate History of Video Games*. Three Rivers Press. ISBN 0-7615-3643-4.

Szeliski, R. (2010). *Computer Vision: Algorithms and Applications*. New York: Springer.

Turing A. *Computing Machinery and Intelligence*. Mind LIX (236): d LIX (236): 433–460, doi:10.1093/mind/LIX.236.433 460, doi:10.1093/mind/LIX.236.433

Winston, P. H. (1992). *Artificial Intelligence (Third edition)*. Boston: Addison-Wesley.

Agradecimientos

Autora

Alexandra La Cruz



**viu
.es**