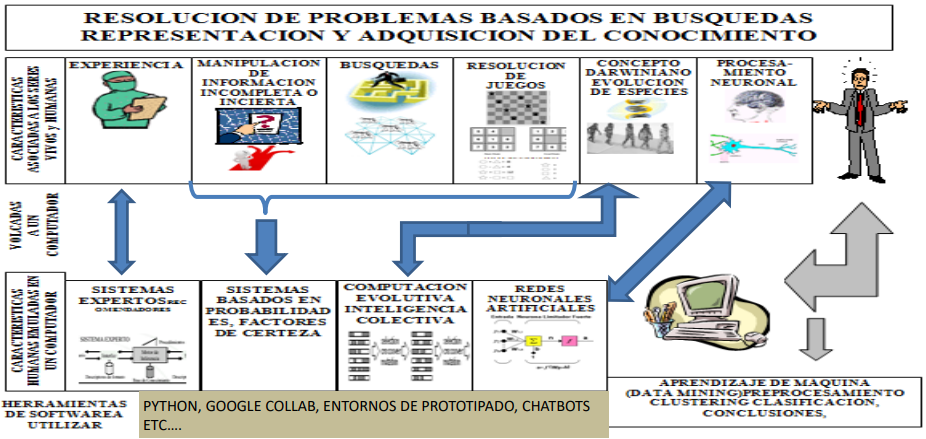
Inteligencia Artificial

CONTROL 1

Unidad 1. - INTRODUCCION

**Concepto de IA**

* Es “el estudio y diseño de agentes inteligentes”, donde un agente inteligente es un sistema que percibe su entorno y toma acciones que maximizan sus posibilidades de éxito.
* Según John McCarthy : “la ciencia y la ingeniería de fabricar máquinas inteligentes”
* La automatización de actividades que vinculamos con procesos de pensamientos humanos, actividades tales como la toma de decisiones, resolución de problemas, aprendizaje
* El arte de crear maquinas con capacidad de realizar funciones , que realizadas por personas requieren inteligencias
* El estudio de las facultades mentales mediante el uso de modelos computacionales
* La rama de la ciencia de la computación que se ocupa de la automatización de la conducta inteligente
* Simulación de la inteligencia humana en maquinas que son programadas para pensar y aprender (RAUL KLENZI)

****

**Objetivo:**

Los problemas (u objetivos) centrales de la investigación de la IA incluyen el razonamiento, el conocimiento, la planificación, el aprendizaje, el procesamiento del lenguaje natural (comunicación), la percepción y la capacidad de mover y manipular objetos.

**Características principales:**

*Evolución*: Debido a los avances en la recuperación y análisis de datos, junto al crecimiento en el poder de procesamiento, se acelero el uso de las IA en organizaciones. Sin embargo, está todavía en una etapa inicial.

*Global:* La capacidad de las maquinas para procesar datos al instante no esta limitada a limites geográficas, es decir, que el análisis y adquisición de datos no tiene fronteras. En consecuencia esto facilita el comercio mundial.

*Multifuncional*: La IA es el medio y la plataforma que puede unificar organizaciones

*Creador de valor*: La innovación y eficiencia generadas por la IA ayudan a las empresas a crear valor en términos de ahorro de costos y tiempo.

*Sensible al tiempo*: La velocidad se ha convertido en la base de la ventaja competitiva. La empresa con la mejor arquitectura de IA esta lista para una ventaja competitiva.

**Hitos históricos y referencias**

1997: Deep Blue de IBM gana al campeón mundial de ajedrez Garry Kasparov, demostrando la capacidad de explorar grandes espacios de búsqueda gracias a avances en hardware.

2011: Watson de IBM gana un juego de preguntas y respuestas, accediendo a múltiples bases de datos y comprendiendo preguntas complejas, incluso con sarcasmo e ironía, aunque la comunicación era textual.

2014: Google DeepMind's AlphaGo gana al campeón mundial de Go, un juego con una complejidad mucho mayor que el ajedrez. Sus estrategias sorprendieron por ser aparentemente ilógicas pero efectivas.

2016: AlphaGo vence a Lee Sedol, el mejor jugador mundial de Go, marcando un hito en la historia de la IA.

**Diferentes conceptos predominantes**



* **Inteligencia Artificial** (IA)es un término acuñado en1956 en la primer reunión de científicos y referentes, que pretendía definir el concepto y posteriormente en el tiempo, automatizar en un entorno de hardware y soft, diferentes aspectos en los que se reconocía inteligencia (percepción, razonamiento, aprendizaje y acción).
* **Inteligencia Computacional** (IC). La IC usa aspectos observables e imitables de la actividad inteligente del ser humano y la naturaleza realizando acciones inteligentes, procesando datos en paralelo, regulando y ajustando estas acciones a través de mecanismos de retroalimentación y generando a su vez una subárea denominada **Machine Learning** (ML).
* Así el ML a través del KDD **( Knowledge Data Discovery - Descubrimiento de Conocimiento en Datos**-) logra, desde la preparación de los datos orientado a la mejora en la performance de los algoritmos de extracción de conocimiento y mejoras constante en las técnicas de visualización del conocimiento generado, conformar el área de **Ciencia de Datos (Data Science DS**) de extraordinaria potencialidad en la medida que permite extraer conocimiento desde la más variada tipología de datos.
* El **Aprendizaje Profundo (DL)** es una nueva área de investigación en Aprendizaje de Máquinas, que se ha introducido con el objetivo de acercar el ML a uno de sus objetivos originales que tenía la IA. Aprendizaje Profundo es sobre el aprendizaje de múltiples niveles de representación y abstracción que ayudan a dar sentido a datos como imágenes, sonido y texto.

**Requisitos de un sistema inteligente**

1. DEBE Habilitado para aprender. El sistema debe mantener una memoria de aprendizaje.
   1. Debe poder aprender de su propia experiencia, compuesta por:
      1. Aprendizaje por prueba y error
      2. Por observación
      3. Mediante razonamiento activo
   2. DEBE estar capacitado para aprender por instrucción. Debe tener programado “instintos”. Esto es un punto de arranque con inteligencia incipiente
2. El sistema DEBE ser autónomo, tener capacidad de hacer cosas por si mismo, esto se puede interpretar como:
   1. Capacitado para afectar su propio medio basado en la independencia de sus propias conclusiones
   2. Debe ser primero su propio maestro y no requerir la intervención del operador para funcionar. No es necesario que reciba ordenes de otro sistema
   3. Debe estar motivado, a través de necesidades que se satisfagan por sus propias acciones
3. El sistema debe tener la capacidad para razonar, o usar de alguna forma de razonamiento basado en su base de conocimiento. DEBE ser capaz de elaborar discernimientos que a posteriori formen parte de su base de conocimiento, reconociendo el grado de certeza del conocimiento generado por lo que debe tener algún manejo de incertidumbre.
4. El sistema DEBE estar capacitado para desarrollar su propia conciencia. Esto esta relacionado con la autonomía, el razonamiento y el aprendizaje, pero también incorpora la necesidad de sensores internos y externos. Sin estos sensores no se aprecia la diferencia entre el “mi” y el “fuera de mi”.

**Aplicaciones varias:**

Vida cotidiana:

* Asistentes virtuales
* Filtrado de correo no deseado
* Recomendaciones de compras en línea

Industria:

* Automatizacion de procesos
* Control de calidad
* Planificacion de la cadena de suministro.

**Implicaciones éticas del uso de la IA**

**Privacidad y seguridad de los datos:**

* **Recopilación y uso de datos:** Los sistemas de IA dependen de **grandes cantidades de datos personales** para funcionar eficazmente, lo que genera **preocupaciones significativas sobre la privacidad** . La recopilación de datos como preferencias de viaje, identificación personal y detalles de pago requiere **medidas estrictas de protección de datos** .
* **Riesgo de filtración de datos:** Industrias como la de viajes son objetivos lucrativos para los ciberataques. Es crucial garantizar **protocolos de ciberseguridad sólidos** para proteger la información confidencial de las infracciones. Existe una **obligación ética de salvaguardar los datos de los clientes** .
* **Transparencia en el manejo de datos:** Los usuarios deben ser informados sobre **qué datos se recopilan, cómo se utilizan y quién tiene acceso a ellos**. Las **políticas de datos transparentes fomentan la confianza** y aseguran el cumplimiento de regulaciones como el Reglamento General de Protección de Datos (GDPR) .

**Sesgo y equidad:**

* **Sesgo algorítmico:** Los **algoritmos de IA pueden perpetuar involuntariamente los sesgos presentes en los datos de entrenamiento**, lo que lleva a un trato injusto de ciertos grupos. Por ejemplo, los algoritmos de fijación de precios podrían discriminar datos demográficos específicos según los niveles de ingresos o la ubicación geográfica.
* **Garantizar la equidad:** Las organizaciones deben **trabajar activamente para identificar y mitigar los sesgos en sus sistemas de IA**. Esto incluye **auditorías periódicas de los algoritmos** e implementar **pautas de equidad** para asegurar un trato justo para todos los usuarios.
* **Diseño inclusivo:** Desarrollar soluciones de IA que satisfagan las **diversas necesidades y orígenes de los usuarios** ayuda a prevenir la discriminación y promueve la inclusión.

**Desplazamiento laboral e impacto en la fuerza laboral:**

* **Automatización de empleos:** La automatización impulsada por la IA puede llevar al **desplazamiento de empleos**, particularmente en roles como servicio al cliente, agentes de reservas y guías turísticos. Esto plantea **preocupaciones éticas sobre el futuro del empleo** en el sector .
* **Reskilling y Upskilling:** Las empresas tienen la **responsabilidad ética de apoyar a su fuerza laboral** a través de iniciativas de *reskilling* (aprendizaje de nuevas habilidades para un nuevo rol) y *upskilling* (aprendizaje de nuevas habilidades para mejorar el desempeño en el rol actual). Ofrecer programas de formación puede ayudar a los empleados a hacer la transición a nuevos roles creados por el avance de la IA.

**Sentido común y lógica**

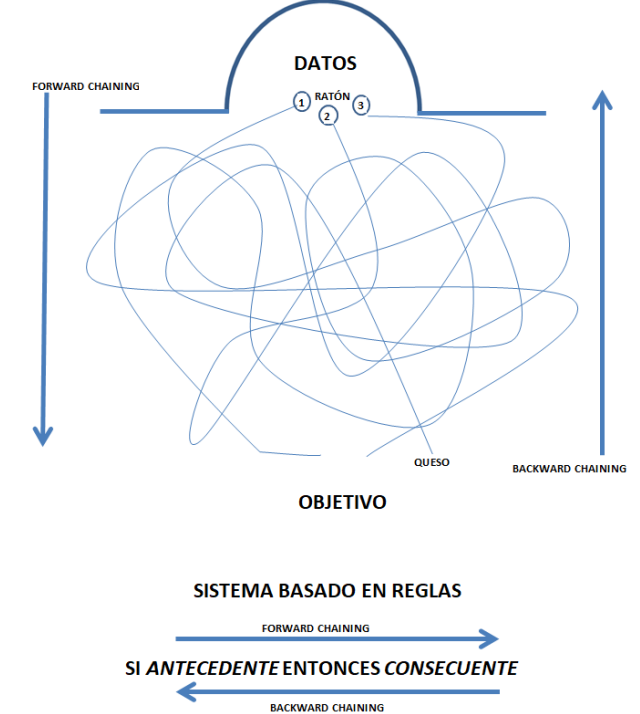
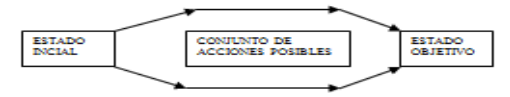
Modus ponens

Si A entonces B DATO: A es verdadero ENTONCES B es verdadero  
  
Modus Tollens  
Si B entonces A DATO: B es falso ENTONCES A es falso

Unidad 1 – **Formalización de conceptos**

| **Dimensión** | **Inteligencia Artificial** | **Programación Convencional** |
| --- | --- | --- |
| **Procesamiento** | Principalmente Simbólico | Principalmente computacional |
| **Tipo de Entrada** | Puede ser incompleta | Debe ser completa |
| **Búsqueda** | Heurística mayoritariamente | Algoritmos |
| **Explicación** | Provee | Usualmente no provee |
| **Interés Fundamental** | Conocimiento | Datos, información |
| **Estructura** | Control separado del conocimiento | Control e información integrados |
| **Naturaleza de la salida** | Puede ser incompleta | Debe ser correcta |
| **Mantenimiento y adaptación** | Relativamente fácil y modular | Usualmente difícil |
| **Capacidad de razonar** | Si tiene | No tiene |

**Definición de problema**

Una persona se enfrenta a un problema cuando desea satisfacer cierto objetivo y no conoce inmediatamente que acción o serie de acciones, debe ejecutar para conseguirlo. La imposibilidad muchas veces es producida por la existencia de multiples formas de alcanzar el objetivo, estas formas pueden afectar la efectividad y eficiencia de la solución. 

**Representacion de un problema**

* **Claridad**: Para facilitar la especificación de los distintos componentes del problema y su interpretación en el contexto adecuado, “hace evidente la relación entre el modelo y el problema real”
* **Exactitud**: El modelo debe representarse en forma precisa y deber ser una representación suficientemente fiel a la realidad como para no distorsionarla
* **Completitud**: El modelo debe representar todos los aspectos pertinentes que intervengan en la solución del problema
* **Eficiencia y Conclusión**: La representación se debe usar en forma eficiente, las características irrelevantes deben ser omitidas.
* **Utilidad**: El modelo debe sugerir un buen método para resolver el problema

**Representación basada en reglas**

**SI condición ENTONCES acción**

Propiedades de las reglas: PRECISION, CONSISTENCIA, COMPLETITUD E INDEPENDENCIA

**Métodos de inferencia**: Una regla tiene dos partes, la premisa y la conclusión. El proceso de inferencia consiste en crear cadenas de reglas hasta que la premisa de cada regla en la cadena iguale la conclusión de otra regla, o la base de hechos.

Hay dos aproximaciones diferentes y fundamentales para controlar la inferencia con reglas:

1. **Forward chaining** (Encadenamiento hacia adelante) Comenzamos con la base de hechos y encontramos reglas con premisas que igualan los hechos. Esto es, vamos desde los hechos hacia el objetivo.
2. **Backward chaining** (Encadenamiento hacia atrás) Comenzamos con el objetivo y encontramos reglas cuyas conclusiones coinciden con el objetivo, tomando la

premisa de aquella regla como nuevo subobjetivo.

**Factor de ramificación**

El factor de ramificación es el numero de **b** estados que se producen al expandir el estado dentro de un espacio de estados, se consideran los siguientes parámetros:

* **b** factor de ramificación
* **d** profundidad de la solución
* **m** profundidad máxima del árbol de búsqueda
* **l (L)** limite de profundidad
* **Memoria necesaria : b^d**

# BÚSQUEDA DE COSTO UNIFORME

Se modifica la estrategia preferentemente por amplitud en el sentido de expandir siempre el nodo de menor costo en el margen (medido por el costo de la ruta g(n) en vez del nodo de menor profundidad).

# BÚSQUEDA PREFERENTEMENTE POR PROFUNDIDAD

Se expanden los nodos que se encuentra en lo más profundo del árbol. Cuando la búsqueda conduce aún callejón sin salida (un nodo sin meta que no tiene expansión) se revierte la búsqueda y se expanden los nodos de niveles menos profundos. La implementación utiliza una lista de espera que va poniendo los estados recién generados a la cabeza de tal lista. Puesto que el nodo expandido fue el más profundo, los respectivos sucesores estarán a profundidades cada vez mayores. El volumen de memoria necesario es bastante modesto. Sólo es necesario guardar la ruta que vaya del nodo raíz al nodo hoja, junto con los nodos restantes no expandidos por cada nodo de la ruta.

**Cantidad de memoria necesaria será m\* b**

# BÚSQUEDA LIMITADA POR PROFUNDIDAD

Se eliminan las dificultades que conlleva la búsqueda preferentemente por profundidad al imponer un límite a la profundidad máxima de una ruta. Es completa, pero no óptima.

**CLIPS**

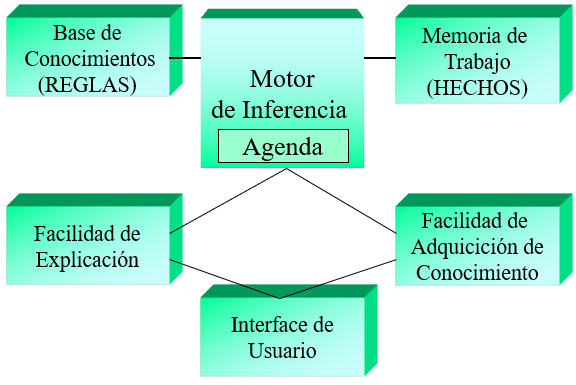
CLIPS (C Language Integrated Production System) es una herramienta para desarrollar sistemas expertos. Es un sistema de producción con encadenamiento hacia adelante.

El motor de inferencia incluye un sistema de mantenimiento de verdad, adición dinámica de reglas, y diferentes estrategias de resolución de conflictos.

**Sistema experto**

Es un sistema informático que emula el razonamiento humano actuando tal y como lo haría un experto en un área de conocimiento.

**Estructura clásica de un sistema experto**



* **Interface de usuario**: permite la comunicación entre el usuario y el SE
* **Facilidad de explicación**: explica al usuario el razonamiento del SE
* **Memoria de trabajo**: Una base de datos global con los hechos usados por las reglas.
* **Motor de inferencia (MI)**: hace inferencias para decidir cuales reglas se satisfacen, da prioridad a las reglas satisfechas, selecciona la regla a aplicar y ejecuta la regla.
* **Agenda**: Lista de reclas creadas por el MI, cuyos patrones están satisfechos por hechos de la MT, contiene las reglas **activadas** que aun **no se han ejecutado**
* **Facilidad de adquisición de conocimiento:** via por la que el usuario introduceconocimiento
* **Base de conocimiento (BC):** alguna forma de representación del conocimiento experto humano

**BC**

* Conocimiento heurístico: Base a reglas
* Conocimiento de Control: Modulos, reglas, instrucciones embebidas

**Base de hechos**

* Hechos: ordenados, no ordenados (templates), objetos (COOL)
* Patrones: Lista de hechos

**Motor de inferencia**

* Encadenamiento hacia adelante
* Selección y ejecución de reglas
* Resolución de conflictos

**Tipos de datos (8 tipos de datos primitivos)**

* Float
* Integer
* Symbol
* String
* Direcciones externas
* Direcciones de hechos
* Nombres de instancias
* Direcciones de instancias
* Lexema (mezcla de symbol y string)

**Hechos** **(facts)**

Son patrones que pueden tener uno o varios campos, el primero suele representar una relación entre los restantes (comienzan con un simbolo), pueden mezclar varios tipos de datos.

Se añaden a una memoria de trabajo con el comando assert y se los elimina con el comando RETRACT

Todos los hechos se almacenan en la lista de hechos

A cada hecho se le asigna un identificador UNICO

**Ordenes básicas**

* (assert <hecho>): agrega o afirma un hecho.
* (retract <dirección de hecho>): elimina un hecho
* (facts): muestra la lista de hechos de la memoria de trabajo.
* (clear): elimina todos hechos y construcciones de la memoria de trabajo.
* (reset): elimina los hechos de la memoria de trabajo, elimina las activaciones de la agenda y restaura las condiciones iniciales.
* (exit): sale de CLIPS.
* (run): comienza la ejecución de un progama CLIPS.

**Definición de Hechos no Ordenados**

Representacion de conocimiento mediante templates

**Sintaxis:**

*(deftemplate <nombre> [<comentario>] <definición de slot>\*)*

*<definición de slot>::= <slot simple> | <multislot>*

*<slot simple>::= (slot <nombre> <atributos template>)*

*<multislot>::= (multislot <nombre> <atributos template>)*

*<atributos template>::= <atributo default> | <restricciones atributo>*

**Constructor deffacts**

Permite que una lista de hechos sean automáticamente afirmados cada vez que se ejecute el comando **reset**

(Definir hechos iniciales)

**Sintaxis:**

*(deffacts <nombre> [<comentario>] <hecho>\*)*

* **Hechos ordenados:**
  + Ventaja: Se pueden utilizar sin declaración previa
  + Inconveniente: Ningún control sobre el tipo de datos
  + Inconveniente: poco explícito (fuente potencial de errores)
  + Atención: ¡La posición de un valor puede tener importancia!
    - **(empleado “Fernández” “Juan” 27 planta permanente)**
    - **(empleado “Juan” “Fernández” 27 planta permanente)**
* **Hechos estructurados:**
  + Ventaja: Mucho más explícitos que los hechos ordenados
  + Ventaja: Control de los tipos de datos
  + Inconveniente: Precisa de declaración previa

**Reglas**

Una regla es un colección de condiciones y acciones que se toman cuando las condiciones se satisfacen. Representan una parte de conocimiento sobre la solución del problema.

Las reglas se ejecutan (o **disparan**) basadas en la existencia de hechos o instancias de clases definidas por el usuario.

El **motor de inferencia** es el encargado de intentar emparejar reglas con el estado actual del sistema (hechos, instancias) y aplicar las acciones

Una regla en CLIPS consta de un nombre, un antecedente o parte izquierda (LHS) y un consecuente o parte derecha de la regla (RHS).La LHS de una regla está compuesta por un conjunto de elementos condicionales (CEs) que son típicamente patrones. La RHS de una regla consiste de un conjunto de acciones que se ejecutan secuencialmente si y solo si los CEs de la regla se satisfacen.

La LHS y la RHS están separadas por una flecha (=>).

**Sintaxis:**

*(defrule <nombre> [<comentario>]*

*[<declaración>] ; Propiedades de la regla*

*<elemento condicional>\* ; Parte izquierda de la regla (LHS)*

*=>*

*<acción>\*) ; Parte derecha de la regla (RHS)*

**Patrones**

Hasta ahora, los patrones usados para hacer coincidir reglas con hechos han sido muy simples y restrictivos. Cada patrón coincidía con un hecho específico.

Mediante el uso de comodines, es posible hacer que las reglas coincidan con múltiples hechos, ejecutando sus acciones repetidamente.

Los comodines que dispone CLIPS son:

? Comodín para campos simples, puede reemplazar cualquier símbolo, empareja con cualquier símbolo.

$? Comodín para multicampos, empareja cero o más símbolos

**Variables**

Las variables son indispensables. Si usamos algo como ?var en lugar de solo ?, podríamos usar el valor de ?var cada vez que la regla donde está definida, sea disparada.

?x es una variable que empareja con cualquier símbolo, guardando su valor.

$?x es una variable que empareja con cero o más símbolos, guardando sus valores.

**Motor de inferencia**

El motor de inferencia trata de emparejar la lista de hechos con los patrones de las reglas. Si todos los patrones de una regla están emparejados se dice que dicha regla está **activada**.

Las activaciones se mantienen en la **agenda**, en que se disponen por orden de prioridad. Para insertar una activación en la **agenda**, se siguen las **estrategias de resolución de conflictos**

**Estrategias de resolución**

Cuando una regla es activada, se coloca en la agenda según los siguientes criterios:

1. Las reglas más recientemente activadas se colocan encima de las reglas con menor prioridad, y debajo de las de mayor prioridad.

2. Entre reglas de la misma prioridad, se emplea la **estrategia de resolución de conflictos**.

3. Si varias reglas son activadas por la afirmación de los mismos hechos y no se puede determinar su orden en la agenda según los criterios anteriores, se insertan de forma arbitraria

Existen diversos mecanismos entre los mas populares están:

**Resolucion de conflictos**

**Refracción**: una regla no puede dispararse mas de una vez con los mismos hechos (evita ciclos)

**Novedad (recency):** las instancias de reglas que utilizan hechos mas recientes son preferidas a las que utilizan hechos mas viejos

**Especificidad**: instancias derivadas de reglas mas específicas (con mayor número de condiciones, es decir más difíciles de cumplir) son preferidas

**Prioridades (salience):** asociadas a las reglas

**Estrategias de CLIPS**

Estrategia de **simplicidad/complejidad**

El criterio de ordenación es la **especificidad** de la regla, esto es, el número de comparaciones que deben realizarse en el antecedente.

Cada comparación a una constante o una variable previamente ligada agregan uno a la especificidad. Cada llamado a una función en la LHS de una regla como parte de: **:**, **=**, o chequeo de un elemento condicional, agregan uno a la especificidad. Las funciones booleanas and, or y not, no agregan a la especificidad de la regla, pero sus argumentos sí lo hacen.

Estrategia **aleatoria**.

A cada activación se le asigna un número aleatorio para determinar su orden en la agenda. Siempre se asigna el mismo número en diferentes ejecuciones.

Estrategia **Lex (OPS5)**.

Se asocia a cada hecho y ejemplar el tiempo en que fueron creados, y se da mayor prioridad a las reglas con un hecho más reciente, comparando los patrones en orden descendente.

Estrategia **MEA (OPS5)**

Aplicamos la misma estrategia de LEX mirando sólo el primer patrón.

**Hechos ordenados**

Constan de un símbolo seguido por cero o más campos separados por espacios en blancos y delimitados por un paréntesis que abre y uno que cierra.

El primer campo especifica una “relación” que se aplica al resto de los campos.

**Hechos no ordenados**

Se definen con el constructor **deftemplate**.

Permiten definir la estructura de un hecho asignando nombres a cada campo del mismo.

Un template es análogo a un registro o una estructura en lenguajes tales como Pascal y C respectivamente.

Los campos de un template se denominan **slots**.

Los slots pueden tener restricciones de tipo, valor y rango numérico. Pueden tener valores por defecto.

CLIPS obtiene datos del usuario por medio de la función read. El programa espera que el usuario ingrese algo, luego sustituye dicha respuesta.

Variables Temporales: solo existen en el interior de la regla, su valor no se puede usar en ningún otro lugar

**Acciones Procedurales:**

* Bind: cumple la función de ligadura ya que sirve para asignar un valor a una variable
* Estructura condicional,
* estructuras repetitivas,
* Return,
* Break
* Switct
* Funciones definidas por el usuario: tiene cinco elementos:

1. Nombre (un símbolo),
2. Un comentario (opcional),
3. Una lista de cero o más parámetros requeridos (variables simples),
4. Un parámetro comodín opcional que sirve para manejar un numero de variables de argumentos (variable multicampo),
5. Una secuencia de acciones o expresiones que serán ejecutadas en orden cuando la función sea llamada

El valor devuelto de la función es la última acción y por defecto es False

Variables Globales: Son variables que pueden ser usadas por mas de una regla o función sin perder sus valores, se declaran usando defglobal

**Manipulación de valores Multicampo:**

* Créate: crea un valor multicampo con las expresiones dadas como parámetros, devuelve el valor creado.
* Delete: borra todos los campos en el rango especificado de la expresión multicampo y devuelve el resultado
* Explode: Devuelve el valor multicampo creado a partir de los campos contenidos en el string
* First: Devuelve el primer campo de una expresión multicampo
* Implode: Devuelve un string que contiene los campos del valor multicampo especificado
* Insert: inserta todos los valores simples o de multicampo en la expresión de multicampo antes del n-ésimo valor (definido por la expresión entera) de la expresión multicampo dada.
* Length: devuelve el número de campos del parámetro dado.
* Member: devuelve la posición del primer argumento en el segundo, o False si el primer argumento no está en el segundo.
* Nth: devuelve el n-ésimo campo (dado por la expresión entera
* Replace: reemplaza los campos en el rango especificado por <inicio> y <final> en la expresión multicampo con todos los del último argumento y devuelve el resultado.
* Rest: devuelve un valor multicampo con todos los elementos originales excepto el primero.
* Subseq: extrae los campos en el rango <inicio>..<final> del primer argumento, y los devuelve en el valor multicampo

**Restricciones en Patrones**

* Restricciones conectivas: ~(not) ; &(and); |(or)

Mediante el uso de estos símbolos el programador puede condicionar un campo para que cumpla ciertas condiciones. & elemento de conexión entre la variable de campo y sus restricciones

* Restricciones con predicados: (p ? ?y&:(oddp ¿y))

Permiten restringir un campo a la verdad de una determinada expresión booleana. Son útiles para realizar pruebas de predicado directamente dentro de los patrones. Es similar a realizar un **test** luego de un patrón.

**Sintaxis:** se usan los dos puntos ‘**:**’seguido de la expresión booleana.

Normalmente va acompañada de restricciones conectivas tipo AND (&), también puede usarse con los conectivos OR (|), y NOT (~).

* Restricciones con valor retornado: (p ?x ?y& =(\* 2 ?x))

Permite que el valor devuelto por una función se utilice para hacer comparaciones dentro de un patrón.

Puede utilizarse junto con las restricciones de campo conectivo &, ~ y |.

**Sintaxis:** Signo igual ‘**=**’ seguido por la función a evaluar

**Funciones de Predicado**

Se define como cualquier función que devuelve el símbolo TRUE o el símbolo FALSE. CLIPS trata cualquier valor diferente al símbolo FALSE como TRUE.

Las funciones de predicado pueden ser funciones predefinidas o definidas por el usuario

* Predicados de tipo
* Predicados de comparación
* Predicados booleanos

**Restricciones test, exists y forall**

* ***test*** *(test <llamada-a-función>)*

Se usa para evaluar expresiones en EC de una regla. El cual se satisface si la llamada a la función que aparezca dentro de él devuelve cualquier valor distinto de FALSE. En caso contrario el EC no se satisface

* ***exists*** *(exists <elemento-condicional>)*

Este EC permite que se produzca un *pattern matching* cuando al menos exista un hecho que satisfaga la regla, sin tener en cuenta el número total de hechos que pudiesen *matchear*.

* ***forall*** *(forall <primer-EC> <resto-de-ECs>)*

Permite el matching basado en un conjunto de EC´s que son satisfechos por cada ocurrencia de otro EC. Para que se satisfaga, todo lo que machee con el primer-EC debe tener hechos que macheen con los demás CE.

Restricciones los slots

**Restricciones de tipo**: atributo **type**

Define el tipo de datos que puede tomar un slot. Los tipos válidos son SYMBOL, STRING, LEXEME, INTEGER, FLOAT, NUMBER

Si se especifica ?VARIABLE, el slot puede tomar cualquier tipo de dato

LEXEME, equivale a SYMBOL y STRING. NUMBER, equivale a INTEGER y FLOAT

**Restricciones de valores permitidos**: atributo **allowed-**

Especifica los valores concretos permitidos para un tipo específico.

Existen siete atributos de esta clase: allowed-symbols, allowed-strings, allowed-lexemes, allowed-integers, allowed-floats, allowed-numbers, allowed-values.

?VARIABLE, indica que cualquier valor especificado es legal.

Por defecto, el atributo toma (allowed-values ?VARIABLE)

**Restricciones de rango**: atributo **range**

Permite restringir los valores legales de un **tipo numérico** a un rango determinado. Límite inferior y superior pueden ser ?VARIABLE (menos infinito y mas infinito, respectivamente).

**Restricciones de cardinalidad**: atributo **cadinality**

Permite restringir el número de valores mínimo y máximo que un slot puede contener. Límite inferior y superior pueden ser ?VARIABLE (indica que no hay número mínimo o máximo de valores que el slot deba contener).

CONTROL 2

# 1. Heurísticas

Una **heurística** es un planteamiento de autoestudio que se emplea para llegar a la solución de un problema. A diferencia de un sistema no heurístico, que refleja directamente el conocimiento existente de un algoritmo capaz de dar la solución, un sistema heurístico debe construirse de tal manera que pueda adquirir el conocimiento necesario para obtener la solución requerida. Las técnicas heurísticas se utilizan en Inteligencia Artificial para automatizar tareas para las que no se conoce ningún procedimiento de decisión o como una alternativa más eficaz a un procedimiento de decisión conocido.   
En contraste con la **búsqueda sin información** o **búsqueda ciega**, donde no hay datos sobre el número de pasos o el costo de ruta hacia la meta, la **búsqueda respaldada con información** o **búsqueda heurística** sí puede diferenciar entre caminos alternativos y determinar cuál conduce a la meta.

En el contexto de la búsqueda, una **Función Heurística (h)** se utiliza para calcular estimados de costos. Esta función estima el costo de la ruta más barata que une el estado del nodo actual con un estado meta. Por ejemplo, en problemas de determinación de ruta, una buena función heurística 'h' es la **distancia en línea recta** entre el nodo actual y la ubicación de la meta.

* La **Búsqueda Preferentemente por lo Mejor (BPM)** es una estrategia donde los nodos se ordenan y se expande primero aquel que parece tener la mejor evaluación.
* La **Búsqueda Avara** (Greedy Search) es una BPM que utiliza la función heurística 'h' para escoger el siguiente nodo a expandir, seleccionando siempre el nodo cuyo estado se considera más cercano al estado meta.
* Una **heurística admisible** es aquella función 'h' que nunca sobrestima el costo real para alcanzar la meta.
* Un comportamiento deseable en una heurística es la **monotonicidad**, que ocurre cuando el costo 'f' nunca disminuye a lo largo de las rutas originadas en la raíz. Esto se garantiza al utilizar la **ecuación de la ruta máxima (ERM)**: f(n´) = max(f(n), g(n´)) + h(n´), donde n es el nodo padre de n´.
* **Una buena función 'h'** debe ser eficiente de calcular y precisa para reducir al mínimo la cantidad de nodos expandidos. La **calidad de una heurística** se puede caracterizar mediante el **Factor de Ramificación Efectiva (FRE)**, denotado b\*.
  + Si N es la cantidad total de nodos expandidos por BA\* y d es la profundidad de la solución, b\* satisface N = 1 + b\* + (b\*)² + ... + (b\*)^d. En una heurística bien diseñada, el valor de b\* se aproxima a 1. Se dice que una heurística h2 **domina** a h1 si h2(n) >= h1(n) para todo nodo 'n', *lo cual se traduce directamente en eficiencia.*

*Las funciones heurísticas se pueden inventar de varias maneras, por ejemplo, escribiendo la definición de un problema en lenguaje formal para construir automáticamente un* ***problema relajado*** *(con menos restricciones), cuya solución puede ser una buena heurística para el problema original. También se puede utilizar información estadística obtenida de problemas de adiestramiento. Los* ***rasgos*** *son características de un estado que forman parte de su función de evaluación heurística.*

Las heurísticas también son importantes en los **Problemas que Satisfacen Restricciones (PSR)**. Algunas heurísticas utilizadas en este contexto son:

* La **Heurística de la variable más restringida:** Escoge la variable que tenga el mínimo de valores posibles aún permitidos. Esto tiende a reducir al mínimo el factor de ramificación de la búsqueda. Se utiliza en la verificación anticipada (forward checking).
* La **Heurística de la variable más restrictiva:** Intenta reducir el factor de ramificación futuro asignando un valor a la variable que está relacionada con el mayor número de restricciones con otras variables aún sin valor asignado.
* La **Heurística del valor menos restrictivo:** Escoge un valor para la variable actual que gobierne (restrinja) el mínimo número de valores en variables relacionadas con ella mediante restricciones.
* En algoritmos de mejoramiento iterativo, la **Heurística del mínimo de conflictos** consiste en elegir el valor para una variable que produzca el mínimo de conflictos con otras variables.

*En la programación mediante Inteligencia Artificial, a menudo se utiliza la* ***búsqueda heurística*** *(pasos de solución implícitos), a diferencia de la programación convencional que se basa en algoritmos con pasos de solución explícitos.*

# 2. Búsqueda con Restricciones (Problemas que Satisfacen Restricciones - PSR)

Un **Problema que Satisface Restricciones (PSR)** es un tipo especial de problema que posee propiedades estructurales adicionales a los requisitos básicos de los problemas en general. En los PSR, los **estados** se definen mediante los valores de un conjunto de variables. La **prueba de meta** (o el objetivo) se especifica como un conjunto de restricciones que los valores asignados a las variables deben satisfacer. Cada variable Vi en un PSR tiene un **dominio** Di, que es el conjunto de posibles valores que puede adoptar, pudiendo ser este discreto o continuo.

Para aplicar un algoritmo de búsqueda de propósito general a un PSR, el **estado inicial** suele ser aquel en el que todas las variables todavía no están asignadas. Los **operadores** consisten en asignar un valor a una variable, tomándolo de su conjunto de valores posibles. En la prueba de meta, se verifica si todas las variables están asignadas y si se satisfacen todas las restricciones.

Los PSR se consideran casos especiales de problemas NP completos; sin embargo, en la mayoría de los problemas reales, se puede aprovechar la estructura del problema para eliminar una parte considerable del espacio de búsqueda. Para resolver PSR, se considera confiable emplear la **búsqueda preferentemente por profundidad (BPP)**, con el agregado de una mejora llamada **algoritmo de búsqueda reversiva (de marcha atrás o backtracking)** para intentar otra acción. Una inclusión útil es realizar una prueba antes de la generación de sucesores para verificar si la asignación de una variable es consistente con todas las restricciones de dicha variable en su dominio. La **propagación de restricciones** es un proceso que ocurre, por ejemplo, en la consistencia de arco, donde las opciones se van disminuyendo.

La **Programación Lógica con Restricciones (CLP)** es una alternativa que incorpora un manejo de restricciones con mayor contenido heurístico de poda. En CLP, la ejecución se asemeja a Prolog hasta que encuentra una señal que indica que la búsqueda en un árbol dado puede acelerarse mediante un proceso de poda.

Un ejemplo clásico es el problema de **criptoaritmética** (SEND+MORE=MONEY). Asignar valores a las letras (variables) y verificar si el resultado es correcto mediante prueba y error (**Generate and Test**) es una opción "inocente" que implicaría un árbol de búsqueda de gran tamaño (por ejemplo, 10⁸ potenciales soluciones inicialmente para SEND+MORE=MONEY con 8 letras distintas y 10 posibles valores para cada una). Este árbol podría ser recorrido en profundidad o a lo ancho. Las **restricciones**, como que las letras S y M no pueden ser cero (si son dígitos iniciales) y que las letras deben ser distintas, disminuyen de manera coherente el espacio de búsqueda. Todo lo que disminuya el espacio de búsqueda puede considerarse una restricción, posibilitando encontrar una solución en menor tiempo.

Se describen diferentes algoritmos de búsqueda para PSRs:

* **Generate and Test (GT):** Es el menos restrictivo; las restricciones se comprueban sólo cuando todas las variables tienen valores asignados.
* **Standard Backtracking (SB):** Asigna un valor a la variable actual si esta asignación es consistente con *todas* las asignaciones realizadas anteriormente.
* **Forward Checking (FC):** Asigna un valor a la variable actual si se cumplen las condiciones de SB *y*, además, para cada variable futura, existe al menos un valor en su dominio que es consistente con respecto a todas las demás variables futuras.

El ejemplo de criptoaritmética se analiza con dos alternativas de representación del conocimiento: sin acarreos y con acarreos. La representación **sin acarreos** usa una única ecuación, la restricción alldifferent, y S DISTINTO 0, M DISTINTO 0. El proceso de labeling asigna valores. Esta implementación resolvió el problema en 60ms. La representación **con acarreos** utiliza variables para los acarreos ({0,1}) además de las letras ({0..9}). La estrategia es restringir los dominios de las variables analizando paralelamente un sistema de ecuaciones con un **manejador de restricciones**. Un resolvedor como PLAIR (Partial Look in Ahead Inference Rule) se limita a desechar dominios que no pertenezcan a la intersección. Este enfoque reduce notablemente el tamaño del árbol de búsqueda antes de intentar asignaciones tentativas. La implementación con acarreos demoró 80ms en resolver el ejemplo. En este caso particular, la representación sin acarreos fue más rápida.

Otro ejemplo de problema representado mediante restricciones es el **problema del Pastor, el Lobo, la Oveja y el Repollo**. El estado se representa como una tupla (Pastor, Lobo, Oveja, Repollo) indicando su ubicación en una orilla (Norte o Sur). Las reglas definen las acciones de cruce en bote (con restricciones como no\_come para evitar que el lobo coma la oveja o la oveja coma el repollo si el pastor no está presente).

# 3. Metaheurísticas

Una **METAHEURÍSTICA** es un método heurístico de nivel superior ("meta" significa "más allá") para resolver un tipo de problema computacional general. Utiliza parámetros proporcionados por el usuario sobre procedimientos genéricos y abstractos de una manera que se espera eficiente. Normalmente, estos procedimientos subyacentes son heurísticos.

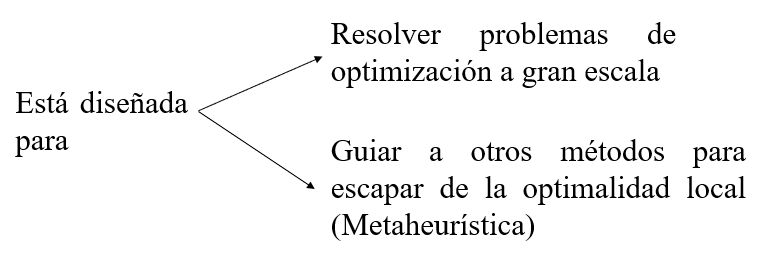
Las metaheurísticas a menudo se derivan de la **teoría de la evolución de Darwin** y la supervivencia del más apto (como los **Algoritmos Genéticos - AG** y la Computación Evolucionaria - CE), la **inteligencia de enjambres**, o **conceptos de memoria**.

Comparadas con otras estrategias de búsqueda:

* Las **búsquedas exhaustivas** (como A lo Ancho y A lo Profundo) recorren la totalidad del espacio de búsqueda hasta encontrar la solución. No usan información inherente al problema. Tienen garantía de encontrar la solución, pero a menudo con un consumo excesivo de tiempo y memoria.
* Las **búsquedas heurísticas** (como la Búsqueda Preferentemente por lo Mejor) recorren la porción más promisoria del espacio de búsqueda conforme al conocimiento inherente al problema (la heurística). Cuanto más conocimiento del problema se tenga, mejor será la heurística. No hay garantía de encontrar la solución, pero si se encuentra, se gasta menos tiempo y memoria.
* Las **Metaheurísticas** se sitúan conceptualmente entre estos dos extremos. Se emulan las condiciones de búsqueda A lo Ancho y A lo Profundo utilizando heurísticas asociadas al problema, aplicando diferentes estrategias de optimización. **No hay garantía de encontrar la solución óptima**, pero con ajustes adecuados de parámetros y operadores, se pueden encontrar **buenas soluciones** (no necesariamente la óptima) con una **escasa ocupación de memoria y gasto de tiempo**. Representan un camino hacia buenas soluciones con aceptables consumos de recursos, a diferencia de las óptimas que a menudo son inalcanzables por su alto costo.

## Búsqueda Tabú

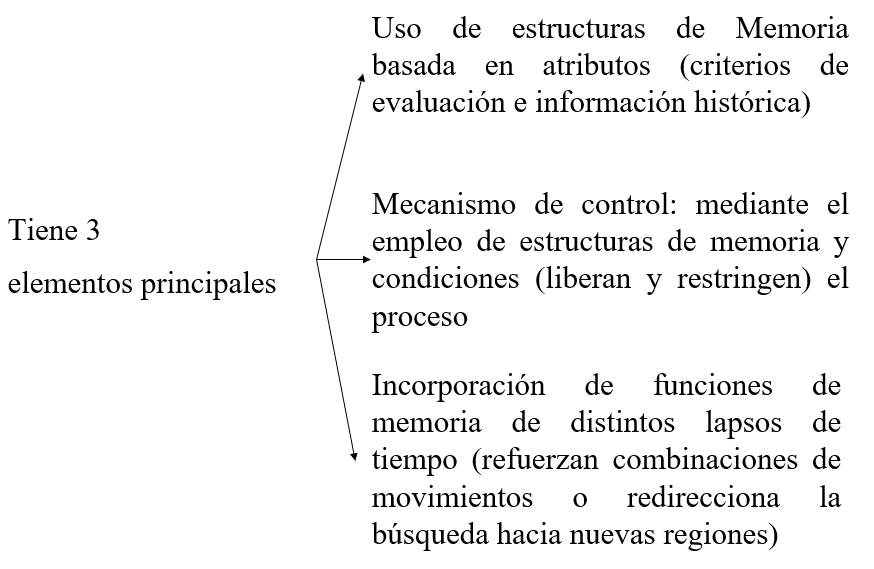
La **Búsqueda Tabú (BT)** es una **metaheurística**. Se clasifica como una clase interesante de **algoritmos heurísticos**. Su objetivo principal es **guiar a la búsqueda local** hacia mejores soluciones.



### Características Clave y Funcionamiento:

1. **Base en la Mejora Paso a Paso:** Está basada en el mejoramiento progresivo de una función de costo al **explorar vecindarios** de soluciones cercanas.
2. **Problema de Óptimos Locales:** Las búsquedas locales "monótonas" (que solo aceptan movimientos que mejoran la solución) pueden quedar **"atrapadas" en óptimos locales**. La Búsqueda Tabú está diseñada para **escapar de la optimalidad local**.
3. **Uso de Memoria:** Se inspira en el uso de la **"Memoria"**, que es la habilidad de usar experiencias pasadas para mejorar la toma de decisiones actual (aprendizaje). Es una técnica de búsqueda determinística sobre un vecindario que **usa memoria (una "lista tabú")**.
4. **La Lista Tabú:** Este es un elemento principal. Su objetivo principal es **impedir "ciclos"** y **prohibir ciertos movimientos**, incluso si mejoran la solución actual, transformando así al optimizador en **global** en lugar de solo local. La lista tabú induce la exploración de **nuevas regiones**. Tiene una longitud (tenencia) y contenido (qué atributos de los movimientos se vuelven tabú).
5. **Siempre Realiza el Mejor Movimiento (No Prohibido):** La BT siempre realiza el mejor movimiento disponible en el vecindario, a menos que esté prohibido por la lista tabú, en cuyo caso busca el mejor movimiento *no tabú* o considera movimientos tabú bajo un criterio de aspiración.
6. **Diversificación e Intensificación:** Emplea memoria para **diversificar** (explorar nuevas regiones) o **intensificar** (enfocarse en regiones prometedoras) la búsqueda.
7. **Determinística:** Aunque pueden existir elementos probabilísticos, la técnica básica es determinística.
8. **Imposición y Relajación de Restricciones:** Sistemáticamente impone y relaja restricciones (a través de la lista tabú y los criterios de aspiración) para permitir la exploración de regiones que de otra manera estarían prohibidas.

### Elementos Principales de un Algoritmo de Búsqueda Tabú:

****

Un procedimiento típico de Búsqueda Tabú incluye:

* Una **codificación** (cómo se representa una solución).
* Una **solución inicial**.
* Una **Función Objetivo** (para evaluar la calidad de una solución).
* Un **operador Movimiento** (para pasar de una solución a otra en el vecindario).
* Una **definición de vecindad**.
* Una o varias **estructuras Tabú (lista(s))**.
* Un **criterio de aspiración** (opcional). Estos permiten anular las restricciones tabú bajo ciertas condiciones, por ejemplo, si un movimiento tabú lleva a una solución mejor que la mejor encontrada hasta ahora.
* Un **criterio de terminación** (cuándo detener la búsqueda).

### Aspectos de la Memoria:

La memoria en la Búsqueda Tabú maneja diferentes aspectos de la historia de la búsqueda:

* **Recencia:** Memoria de corto plazo (cuán recientemente se ha visitado un estado o se ha realizado un movimiento).
* **Frecuencia:** Memoria de largo plazo (cuántas veces se ha visitado un estado o se ha utilizado un atributo de movimiento).
* **Calidad:** Relacionado con la aspiración (qué tan buena es la solución).
* **Influencia:** También relacionado con la aspiración (qué tan lejos se está de un objetivo). (no estudiabamos)

### Componentes

**Lista Tabú**

* **El objetivo principal** de la lista tabú es transformando al optimizador en global en lugar de local. impedir “ciclos”,
* **La función principal** de esta lista es inducir la exploración de nuevas regiones
* **Longitud – fija o dinámica** (generalmente 3,7,10,20, o log2 frecuencia).
* **Contenido** - atributos “desde”, atributos “hacia”; atributos “movimiento”; los mas específicos, los menos restrictivos
* **Frecuencia** - registra soluciones similares a través del contenido tabú . Usualmente se usará para penalizar más que restringir.
* Las restricciones Tabú no son inviolables (criterio de aspiración)

**Vecindario**

* Completo (determinístico)
* Parcial (probabilístico)
* Primer mejora
* Solo los que mejoran

### Criterios de Aspiración

Se introducen en la BT para determinar cuando las restricciones tabú pueden removerse para que pueda aplicarse el movimiento.

**Tipos:**

* **Aspiración por default:** si todos los movimientos posibles son tabú, entonces el movimiento “menos tabú” se selecciona.
* **Aspiración por objetivo:** una aspiración de movimiento se satisface, permitiendo que un movimiento x sea candidato para seleccionarse si F(x)<F(S\*).
* **Aspiración por Dirección de Búsqueda:** Un atributo de aspiración para la solución “s” se satisface si la dirección en “s” proporciona un mejoramiento.

***En resumen****, la Búsqueda Tabú es una metaheurística que, partiendo de una solución inicial, explora el vecindario de soluciones posibles, pero a diferencia de una búsqueda local simple, utiliza una memoria (la lista tabú) para evitar repetir movimientos recientes y quedar atrapada en óptimos locales, buscando así soluciones de mejor calidad en un espacio de búsqueda más amplio. Se destaca por su uso flexible de memoria y su capacidad para imponer y relajar restricciones de forma sistemática.*

## **Algoritmos Genéticos**

Son un ejemplo de metaheurística derivada de la teoría de la evolución. La fuente específica sobre AGs introduce un problema (encontrar la recta más grande dentro de un rectángulo) y menciona que se modela la solución (la recta) definiendo la **estructura del cromosoma**, que es la representación en código de la solución potencial. *Esto ilustra cómo las metaheurísticas utilizan representaciones abstractas (como cromosomas) y procedimientos genéricos para buscar soluciones.*

El ejemplo del **cuadrado mágico de tres por tres** donde se busca una configuración que sume 15 en filas, columnas y diagonales también puede verse como un problema donde se aplican heurísticas (identificar las combinaciones que suman 15, usar la heurística de variable más restringida para posicionar el 5 en el centro). Si bien el texto presenta pasos específicos de colocación, este tipo de problema a menudo es abordado por metaheurísticas cuando el espacio de búsqueda es muy grande y encontrar una solución (o la óptima) es difícil con métodos exactos o heurísticos simples.

# 4. Cuestionario de Búsqueda Tabú

**1. ¿La BT es una heurística o una metaheurística? Justifique**

La Búsqueda Tabú (BT) es una metaheurística que guía a la búsqueda local hacia mejores soluciones. BT utiliza parámetros y procedimientos genéricos de una manera que se espera sea eficiente. Emplea el uso de "Memoria" (un elemento principal) para guiar la búsqueda y escapar de la optimalidad local.

**2. ¿Por qué se dice que la BT es un optimizador global?**

Esto se debe a que utiliza memoria, específicamente una "lista tabú", para prohibir ciertos movimientos, incluso aquellos que mejorarían la solución actual. El objetivo principal de la lista tabú es impedir "ciclos" e inducir la exploración de nuevas regiones del espacio de búsqueda. Al evitar estancarse en óptimos locales, lo que puede ocurrir con búsquedas locales monótonas, la BT aumenta las posibilidades de encontrar la mejor solución global.

**3. ¿Qué tipos de problemas resuelve la BT?**

La Búsqueda Tabú está diseñada para resolver problemas de optimización a gran escala. Se aplica en problemas caracterizados por "Explosión combinatoria", donde una búsqueda exhaustiva es impracticable y las heurísticas tradicionales pueden quedar atrapadas en óptimos locales. La BT se enmarca como una solución de la Inteligencia Artificial para problemas difíciles de resolver como el Problema del Viajante de Comercio (TSP),. En general, es útil en problemas de optimización combinatoria (aquellos que implican determinar una permutación, una combinación, o ambas, de elementos, a menudo sujeto a restricciones) que presentan dificultades para los métodos de búsqueda local básicos.

**4. ¿Cuál es el objetivo de la BT?**

Está diseñada para resolver problemas de optimización a gran escala. Guiar a otros métodos para escapar de la optimalidad local.

**5. ¿Qué tipo de técnica de búsqueda es y sobre qué trabaja?**

Es una metaheurística determinística que utiliza memoria para mejorar la eficiencia de la búsqueda local y evitar caer en óptimos locales. Siempre realiza el mejor movimiento o el mejor movimiento del vecindario. Emplea memoria para diversificar o intensificar la búsqueda

Trabaja sobre:

* Vecindarios de soluciones: explora soluciones cercanas a la actual.
* Memoria (lista tabú): para evitar ciclos y permitir explorar nuevas regiones del espacio de soluciones.
* Problemas de optimización combinatoria, donde busca soluciones óptimas (o casi óptimas) dentro de un gran espacio de búsqueda.

**6. ¿Qué tipos de memoria utiliza y qué almacena cada una?**

La Búsqueda Tabú está inspirada en el uso de la "Memoria", considerada su principal elemento. Esta memoria es flexible y permite usar experiencias pasadas para mejorar la toma de decisiones actual. Utiliza estructuras de memoria basadas en atributos y en información histórica. Los aspectos de la memoria que utiliza son:

* Recencia (Memoria de términos cortos): Almacena información sobre cuán recientemente se ha estado en un lugar o realizado un movimiento.
* Frecuencia (Memoria de términos largos): Registra cuántas veces se han visitado ciertas áreas o soluciones similares.
* Calidad: Relacionada con la aspiración, indica cuán buena está siendo una solución.
* Influencia: Relacionada con la aspiración, indica cuán lejos se está de ciertas áreas.

**7. ¿Qué tipos de búsqueda favorecen las distintas memorias que utiliza?**

La Búsqueda Tabú emplea la memoria para diversificar o intensificar la búsqueda.

La memoria de corto plazo (Recencia), implementada a través de la lista tabú, tiene como objetivo principal impedir ciclos e inducir la exploración de nuevas regiones. Esto favorece la diversificación, guiando la búsqueda hacia áreas no exploradas recientemente.

La memoria de largo plazo (Frecuencia) registra soluciones similares a través del contenido tabú y se usa usualmente para penalizar más que para restringir. Esto también contribuye a la diversificación, desalentando la revisita frecuente de las mismas áreas del espacio de búsqueda

**8. ¿Cómo se llama en BT la búsqueda a lo ancho y la búsqueda a lo profundo y cómo se implementan?**

La BT guía una búsqueda local operando sobre el vecindario de la solución actual. Las metaheurísticas, en general, emulan las condiciones de búsqueda a lo ancho y profundo utilizando estrategias de optimización y heurísticas asociadas al problema. En la BT, la implementación implica:

* Definir una estructura de vecindad para la solución actual. El vecindario puede ser completo (determinístico) o parcial (probabilístico).
* En cada iteración, se selecciona el mejor punto (movimiento) en el vecindario, considerando las restricciones de la lista tabú y el criterio de aspiración.
* Se acepta el movimiento seleccionado (que se convierte en la nueva solución actual).
* Se actualiza la lista tabú.

Este proceso explora el espacio de búsqueda moviéndose de una solución a otra a través de sus vecinos, guiado por la función objetivo y la memoria

**9. ¿En la BT, qué es la lista tabú, cuál es su contenido, su objetivo, su longitud, es inviolable?**

La lista tabú (Tabu\_List - TL) es una estructura de memoria utilizada en la Búsqueda Tabú para prohibir ciertos movimientos.

Contiene atributos de los movimientos o soluciones recientes que se consideran "tabú". Por ejemplo, puede contener "atributos 'desde'", "atributos 'hacia'", o "atributos 'movimiento'"

Su función principal es inducir la exploración de nuevas regiones del espacio de búsqueda.

Su longitud puede ser fija o dinámica. Se suele llamar "Tenure". Los valores habituales para la longitud fija son 3, 7, 10, 20, o calculada como log2 de la frecuencia.

Las restricciones Tabú no son inviolables. Pueden removerse para que un movimiento se aplique si se cumple un criterio de aspiración.

**10. ¿Qué es el criterio de aspiración? Clasifíquelos.**

El criterio de aspiración se introduce en la Búsqueda Tabú para determinar cuándo las restricciones tabú pueden removerse para que un movimiento previamente prohibido pueda aplicarse. Permite aceptar un movimiento tabú si cumple ciertas condiciones.

Clasificación:

* Aspiración por defalut (por defecto)
* Aspiración por objetivo
* Aspiración por Dirección de Búsqueda

**11. Tipos de Criterios de aspiración**

* **Aspiración por defalut (por defecto):** Si todos los movimientos posibles en el vecindario son tabú, entonces se selecciona el movimiento que es "menos tabú".
* **Aspiración por objetivo:** Una aspiración se satisface, permitiendo que un movimiento x sea candidato para seleccionarse si su valor de función objetivo F(x) es **mejor** que el valor de la función objetivo de la **mejor solución encontrada hasta ahora** (F(S\*)). El pseudocódigo específico muestra que un movimiento tabú B se acepta si FO(B) > FO(S\*), lo que implica que S\* es la mejor solución global encontrada hasta el momento y se aplica para problemas de maximización según la notación del pseudocódigo.
* **Aspiración por Dirección de Búsqueda:** Un atributo de aspiración para la solución s se satisface si la dirección en s proporciona un mejoramiento.

**12. ¿En qué se diferencia la BT de Algoritmos genéticos?**

* **BT:** Es una técnica de **búsqueda determinística** (aunque puede tener elementos probabilísticos, como la selección de vecindarios) que opera sobre un **único punto (solución)** en el espacio de búsqueda, explorando su **vecindario**. Su principal elemento es el uso de **memoria** (lista tabú, recencia, frecuencia, etc.) para guiar la búsqueda, **impedir ciclos** y escapar de óptimos locales, actuando como un **optimizador global**. Trabaja con **atributos de movimientos o soluciones** para gestionar la memoria.
* **AG:** Es una metaheurística **bioinspirada** basada en la mecánica de la **selección natural y la genética**. Opera sobre una **población de soluciones (individuos)**. Utiliza **operadores genéticos** como la selección, el cruce (crossover), la mutación y la reinserción para generar nuevas soluciones. Su búsqueda es inherentemente **probabilística**. Trabaja con la **codificación de las soluciones (cromosomas)** y su **fitness** (valor de aptitud). A diferencia de la BT que explora el vecindario de una solución, el AG explora el espacio de búsqueda a través de la manipulación y combinación de individuos en la población.

**13. Realice un análisis comparativo de: BT, AG y ACO teniendo en cuenta sus características, operadores, estructuras, parámetros, problemas que intenta resolver, tipos de búsquedas, etc.**

| **Característica** | **Búsqueda Tabú (BT)** | **Algoritmos Genéticos (AG)** | **Optimización Basada en Colonias de Hormigas (ACO)** |
| --- | --- | --- | --- |
| **Tipo de Metaheurística** | Búsqueda local guiada | Poblacional, Bioinspirada (Evolución) | Poblacional (Multi-agente), Bioinspirada (Hormigas) |
| **Inspiración** | Uso de "Memoria" para mejorar decisiones | Selección natural y genética | Comportamiento de búsqueda de alimento de hormigas |
| **Estructura Principal** | Memoria flexible (Lista Tabú, Recencia, Frecuencia, etc.) | Población de individuos (cromosomas) | Grafo (nodos/arcos), Rastros de Feromona, Información Heurística |
| **Mecanismo de Avance** | Selección del mejor movimiento en el vecindario, respetando lista tabú/aspiración | Generación de nuevas soluciones a través de operadores genéticos (Cruce, Mutación) | Construcción probabilística de soluciones por hormigas |
| **Memoria/Información** | Recencia, Frecuencia, Calidad, Influencia (atributos de movimientos/soluciones) | Población (soluciones pasadas/presentes), Fitness (valor de aptitud) | Feromona (información memorística, histórica), Heurística (información del problema) |
| **Tipo de Búsqueda** | Determinística guiada sobre vecindario | Probabilística, explora el espacio a través de la población | Probabilística (Regla de Transición) |
| **Escape de Óptimos Locales** | Uso de la lista tabú (prohibición de movimientos) y criterio de aspiración | Diversidad en la población, mutación | Evaporación de feromona, regla de transición probabilística, actualización local |
| **Parámetros Clave** | Longitud de la Lista Tabú (Tenure), Definición del vecindario, Criterios de aspiración | Tamaño de la población, Probabilidades de cruce y mutación, Tipo de selección/reinserción | Número de hormigas, Pesos α y β (regla de transición), Tasa de evaporación ρ, Aporte de feromona |
| **Estructuras/Componentes Adicionales** | Función objetivo, Solución inicial, Operador Movimiento, Definición de vecindad, Criterio de terminación | Selección, Crossover, Mutación, Reinserción, Terminación | Lista de nodos visitados (por hormiga), Regla de transición, Actualización de feromona (global/local), Criterio de parada |
| **Problemas que Intenta Resolver** | Optimización a gran escala, guiar otros métodos, escapar de óptimos locales | Problemas de optimización (combinatoria, etc.) | Problemas de optimización representables como grafos con pesos (TSP, Enrutamiento, etc.) |
| **Hibridación Común** | Puede guiar otros métodos | Con búsqueda local (Algoritmos Meméticos) | Con búsqueda local |
| **Soluciones "Élite"** | Sigue la mejor solución encontrada hasta ahora (S\*). Usa S\* en aspiración. | Métodos de Selección y Reinserción Élite, Algoritmos Elitistas. | Algoritmos Elitistas (SHE, SCH, SHMM) refuerzan arcos de la mejor solución. Sigue Mejor\_Global. |

**14. ¿Qué papel cumple la función objetivo en la BT?** La función objetivo (F o FO) es uno de los **componentes principales** de la Búsqueda Tabú. Su papel es fundamental para **evaluar la calidad** de una solución actual (S) o de un movimiento potencial en el vecindario (B). Es utilizada en el pseudocódigo para **seleccionar el "mejor punto"** (B) en el vecindario de la solución actual. Además, la función objetivo es crucial en el **criterio de aspiración por objetivo**, permitiendo que un movimiento tabú sea aceptado si mejora el valor de la mejor solución encontrada hasta ahora (FO(B) > FO(S\*)).

**15. ¿Cómo se elige la solución inicial en la BT?**

La solución inicial (S) es un **componente** de la Búsqueda Tabú. Puede ser elegida de dos maneras:

* **Aleatoria (Random)**.
* **Constructiva (Constructive)**.

**16. ¿Cómo se selecciona el vecindario en la BT?** El vecindario (N(S)) de una solución S es un **componente** de la Búsqueda Tabú. Se define como un mapeo N:S -> 2^S que para cada solución i define un conjunto Si de soluciones "cercanas" a i. En cada iteración, el algoritmo selecciona el **"mejor punto B en el vecindario de S: N(S)"**. La selección del mejor punto se basa en la función objetivo. La estructura del vecindario puede ser **Completa (determinística)** o **Parcial (probabilística)**.

**17. ¿Cuándo se detiene la búsqueda?**

La búsqueda en la Búsqueda Tabú se detiene cuando se alcanza un **criterio de terminación**. El pseudocódigo del algoritmo indica que el bucle principal repeat...until continúa hasta que se cumpla esta condición.

**18. ¿Cómo se selecciona el criterio de terminación en la BT?**

El criterio de terminación es listado como un **componente** de la Búsqueda Tabú. Las fuentes sobre BT no detallan específicamente *cómo* se selecciona este criterio o qué tipos son los más comunes para BT. Sin embargo, en el contexto general de metaheurísticas (como se ve en las fuentes sobre AG), los criterios de terminación pueden basarse en el número de generaciones, tiempo de ejecución, estancamiento del fitness, o alcanzar un umbral de fitness deseado. El pseudocódigo de BT simplemente hace referencia a "alcanzar el criterio de terminación".

**19. ¿Qué se entiende por Recencia (novedad) en BT?**

La Recencia (Recency) es uno de los aspectos de la **memoria** que utiliza la Búsqueda Tabú. Se refiere a la **memoria de términos cortos**. Indica **"Cuan recientemente he estado aquí?"**. Este tipo de memoria se implementa a través de la **lista tabú** y tiene como objetivo principal **impedir ciclos** e **inducir la exploración de nuevas regiones**.

**20. ¿Cómo se calcula el valor de un movimiento?**

Las fuentes no proporcionan una fórmula única para calcular un "valor" genérico de un movimiento. Sin embargo, el proceso de evaluación y selección de un movimiento en la BT implica considerar:

* El valor de la **función objetivo** de la solución resultante del movimiento (FO(B)). El algoritmo busca el "mejor punto B" en el vecindario, lo que implica evaluarlos usando la función objetivo.
* Si el movimiento propuesto (B) está en la **Lista Tabú** (B ∉ TL).
* Si, estando en la Lista Tabú, satisface el **criterio de aspiración** (FO(B) > FO(S\*) en el caso de aspiración por objetivo).

El "valor" de un movimiento, en el contexto de la selección, está dado por su impacto en la función objetivo, su estado tabú y si cumple alguna condición de aspiración.

**21. ¿Cuándo un movimiento es admisible en BT?**

Un movimiento (B) desde el vecindario de la solución actual (S) es **admisible** si cumple una de las siguientes condiciones:

1. **No está en la Lista Tabú** (B ∉ TL).
2. **Está en la Lista Tabú**, pero satisface un **criterio de aspiración**. El pseudocódigo muestra un ejemplo de criterio de aspiración: si FO(B) es mejor que la mejor solución encontrada hasta ahora (FO(S\*)).

**22. ¿Qué se entiende por mejor movimiento en BT?**

El "mejor movimiento" se refiere al **"mejor punto B en el vecindario de S: N(S)"**. Este "mejor punto" es seleccionado inicialmente basándose en la **función objetivo**. Una vez identificado el mejor punto potencial en el vecindario según la función objetivo, se verifica si este movimiento es **admisible** (es decir, no es tabú o cumple el criterio de aspiración si es tabú).

**23. ¿En las estrategias de intensificación y diversificación qué memorias se utilizan?**

La Búsqueda Tabú emplea memoria para **diversificar o intensificar** la búsqueda.

* Para la **diversificación**, se utiliza principalmente la **memoria de corto plazo (Recencia)** implementada por la lista tabú, cuyo objetivo es inducir la exploración de nuevas regiones y evitar ciclos. La **memoria de largo plazo (Frecuencia)**, que registra soluciones similares a través del contenido tabú, también contribuye a la diversificación al penalizar la revisita frecuente de áreas ya exploradas.
* Para la **intensificación**, se utiliza la memoria asociada a la **calidad** y la **influencia** (relacionadas con el criterio de aspiración). El **criterio de aspiración** permite aceptar movimientos tabú si llevan a soluciones significativamente mejores (como FO(B) > FO(S\*)), lo que facilita la intensificación en torno a soluciones prometedoras. Además, las funciones de memoria de distintos lapsos de tiempo pueden **reforzar combinaciones de movimientos**, lo cual apoya la intensificación.

**24. ¿Qué son las soluciones élites y en donde se almacenan?**

Se mantiene un registro de la mejor solución encontrada hasta ahora (S\*), que es análogo al concepto de mejor solución global. Está mejor solución (S\*) se utiliza en el criterio de aspiración por objetivo.

**25. Enumere y describa algunas restricciones tabúes.**

Las restricciones tabú en la Búsqueda Tabú **no son inviolables**. Se implementan a través de la **Lista Tabú**. El contenido de la lista tabú son **atributos de los movimientos** o soluciones recientes que se desean prohibir temporalmente.

* **Atributos "desde":** Un atributo del estado o solución *desde el cual* se realizó un movimiento.
* **Atributos "hacia":** Un atributo del estado o solución *hacia el cual* se realizó un movimiento.
* **Atributos "movimiento":** Atributos que describen la naturaleza del movimiento en sí mismo (por ejemplo, intercambiar la posición de los elementos X e Y).