
UNIVERSIDAD SANTO TOMÁS
FACIENTES VERITATEM
UNIVERSIDAD
SANTO TOMÁS

# PRESENTACIÓN

Ingeniero Electrónico,  Magister en Ingeniería con énfasis en electrónica y estudiante del doctorado en ingeniería con énfasis en eléctrica y electrónica de la UDFJC

Diego Alejandro Barragán Vargas

Docente de electrónica Universidad Santo Tomás de Aquino

Enlace de Interés:
https://scholar.google.com/citations?hl=es&user=Bp3QMQMAAAAJ

UNIVERSIDAD SANTO TOMÁS

# Sesión 6-Árboles de Búsqueda

4 de Septiembre, Bogotá D.C.

# CONTENIDO

## TEXTO COMPLEMENTARIO

Árboles de Búsqueda Informada

Heurísticas

Greedy

Search

A* (Consistencia y Admisibilidad)

# Algoritmos de Búsqueda Informada

**Los algoritmos de búsqueda informada en IA son métodos de búsqueda que utilizan conocimiento adicional, llamado heurística, para priorizar las rutas a explorar.**

CARACTERÍSTICAS

**Al estimar la proximidad de cada paso al objetivo, estos algoritmos pueden encontrar soluciones con mayor rapidez y eficiencia que la búsqueda desinformada o a ciegas.**

**Se utilizan ampliamente en IA para tareas como la búsqueda de rutas y la resolución de acertijos, ya que ayudan a navegar por espacios de búsqueda amplios y complejos.**

**Estos algoritmos priorizan las rutas que parecen más prometedoras hacia el objetivo, usando una función de evaluación para estimar el costo de llegar al estado deseado.**

# Algoritmos Heurísticos

**Un algoritmo heurístico es un método de búsqueda o solución de problemas que no garantiza encontrar la solución óptima, pero sí proporciona una solución suficientemente buena en un tiempo razonable.**

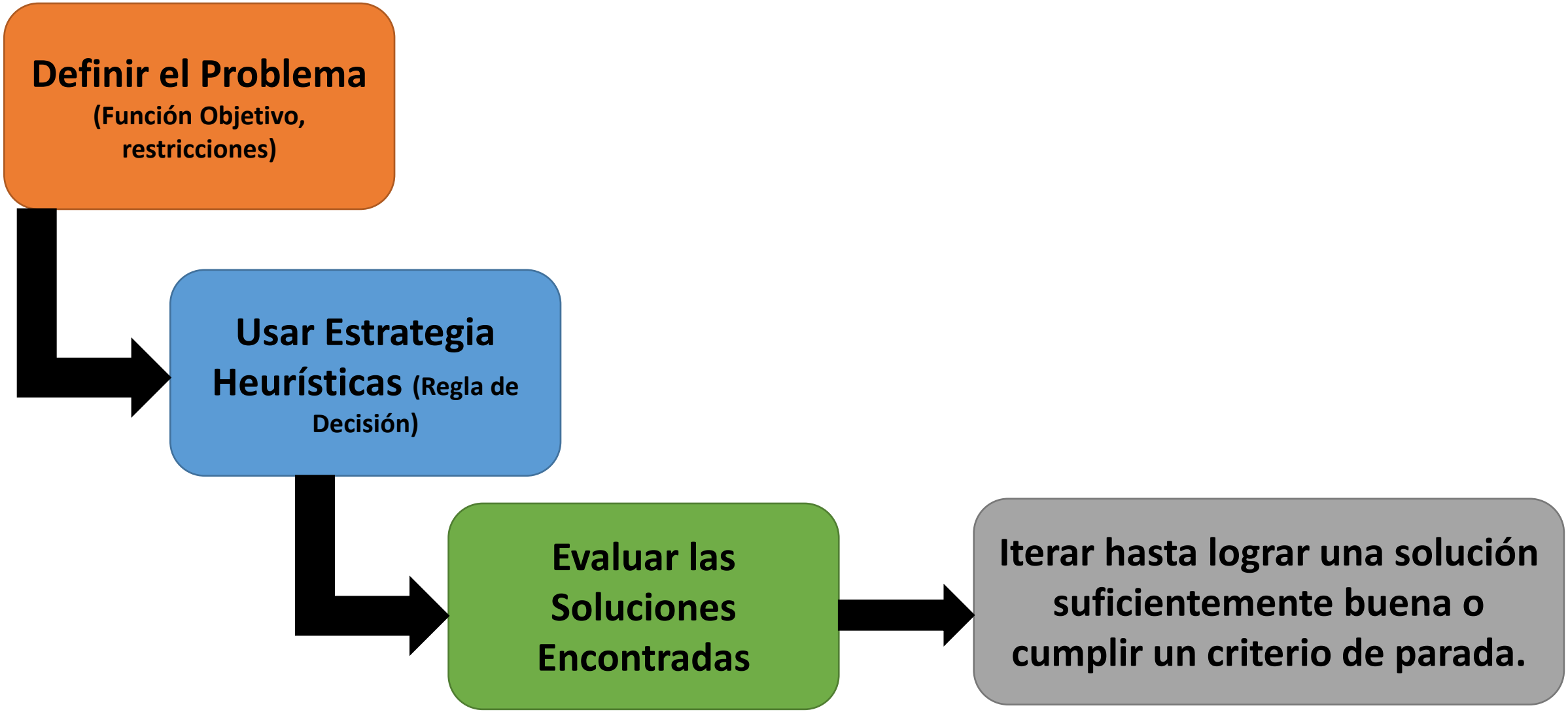
**Se usan cuando:**

- **El espacio de búsqueda es demasiado grande.**
- **Los algoritmos exactos serían muy lentos o inviables.**
- **Es preferible una buena solución rápida a una solución óptima muy costosa.**

**Características:**

| | |
|---|---|
| **Aproximación** | **No siempre encuentran la mejor solución, pero logran soluciones de calidad aceptable.** |
| **Rapidez:** | **Reducen el tiempo de cómputo comparado con algoritmos exactos.** |
| **Flexibilidad:** | **Se adaptan a distintos problemas.** |
| **Basados en reglas de experiencia** | **Usan conocimiento previo, intuición o reglas empíricas.** |
| **Escalabilidad** | **Funcionan bien en problemas grandes donde otros algoritmos colapsarían.** |

## Funcionamiento



# Tipos de Algoritmos Heurísticos

## Búsqueda local

Exploran soluciones cercanas a la actual.

Ejemplo: Hill Climbing (ascenso de colina), Simulated Annealing (enfriamiento simulado).

## Ejemplo: Búsqueda Local – Hill Climbing

```python
import random

def f(x):
    return -(x-3)**2 + 9  # Máximo en x=3

def hill_climbing(iteraciones=1000):
    x = random.uniform(-10, 10)  # punto inicial aleatorio
    for _ in range(iteraciones):
        nuevo_x = x + random.uniform(-1, 1)  # pequeña variación
        if f(nuevo_x) > f(x):
            x = nuevo_x
    return x, f(x)
```

Maximizar la función

$$f(x) = -(x-3)^2 + 9.$$

Encuentra un valor cercano a x=3 aunque no siempre exacto.

# Metaheurísticas

Estrategias más generales que combinan heurísticas simples.

Ejemplo: Algoritmos genéticos, Colonia de hormigas, Enjambre de partículas.

## Algoritmo Genético sencillo

```python
import random

def fitness(x):
    return x**2

def mutacion(x):
    return x ^ (1 << random.randint(0, 4))  # muta un bit

def algoritmo_genetico(generaciones=50, poblacion_size=6):
    poblacion = [random.randint(0, 31) for _ in range(poblacion_size)]
    for _ in range(generaciones):
        poblacion = sorted(poblacion, key=fitness, reverse=True)
        nueva_poblacion = poblacion[:2]  # elitismo
        while len(nueva_poblacion) < poblacion_size:
            padre = random.choice(poblacion[:3])
            hijo = mutacion(padre)
            nueva_poblacion.append(hijo)
        poblacion = nueva_poblacion
    mejor = max(poblacion, key=fitness)
    return mejor, fitness(mejor)

sol = algoritmo_genetico()
print("Mejor solución:", sol)
```

Maximizar la función

$$f(x) = x^2 \text{ en } [0, 31].$$

El algoritmo evoluciona soluciones y encuentra valores cercanos al óptimo

## Greedy (voraces)

Toman decisiones locales óptimas esperando llegar a una solución global aceptable.

Ejemplo: Algoritmo de Dijkstra (camino más corto).

## Algoritmo Greedy (Cambio de monedas)

```python
def cambio_monedas(cantidad, monedas=[50, 20, 10, 5, 1]):
    resultado = []
    for moneda in monedas:
        while cantidad >= moneda:
            cantidad -= moneda
            resultado.append(moneda)
    return resultado

print("Cambio para 87:", cambio_monedas(87))
```

Elige siempre la moneda más grande posible primero (decisión voraz).

Para 87.

**87 → [50, 20, 10, 5, 1, 1]**

| Características | Heurísticas clásicas | Metaheurísticas |
|---|---|---|
| Idea principal | Reglas simples/greedy o mejoras locales que usan conocimiento directo del problema. | Estrategias generales inspiradas en procesos naturales/societales que exploran el espacio de búsqueda combinando exploración y explotación. |
| Alcance de búsqueda | Principalmente local (p. ej. greedy, hill-climbing). | Típicamente global (buscan escapar de óptimos locales). |
| Exploración vs Explotación | Fuerte explotación (decisiones locales); poca exploración. | Balance explícito entre exploración y explotación (mecanismos de diversidad). |
| Garantía de óptimo | No garantizan óptimo global; pueden ser deterministas y converger rápido. | Tampoco garantizan óptimo global, pero tienen mayor probabilidad de encontrar soluciones cercanas al global. |
| Uso de memoria | Baja (sin memoria o mínima). | Pueden usar memoria (p. ej. feromonas en ACO, mejores soluciones históricas en PSO, poblaciones en GA). |
| Operadores típicos | Reglas voraces, intercambio local, evaluación de vecinos. | Crossover/mutación (GA), movimiento por velocidad (PSO), feromonas & probabilidad (ACO), selección de fuentes (ABC), etc. |
| Complejidad computacional | Baja por iteración; muchas iteraciones pueden ser rápidas. | Más costosas por iteración (poblaciones/colonos/enjambres); pero convergen mejor en problemas grandes. |
| Ventajas | Fáciles de entender/implementar; rápidas para soluciones aceptables. | Flexibles, robustas para problemas complejos, adaptables a distintos dominios. |
| Desventajas | Se atascan en óptimos locales; dependen mucho de la solución inicial. | Requieren ajuste de hiperparámetros; mayor coste computacional. |
| Aplicaciones típicas | Asignación simple, heurísticas de construcción (ej. knapsack approximations), greedy para scheduling. | Optimización combinatoria y continua (TSP, diseño, redes, machine learning hyperparam tuning, planificación). |
| Ejemplos | Hill Climbing, Greedy, Búsqueda en primer mejor. | Algoritmos Genéticos (GA), Particle Swarm Optimization (PSO), Ant Colony Optimization (ACO), Artificial Bee Colony (ABC). |

# Algoritmo Genético sencillo

Es una técnica de optimización y búsqueda basada en la evolución natural. Simula cómo los individuos (soluciones) evolucionan generación tras generación aplicando selección, cruce (crossover) y mutación para encontrar soluciones cada vez mejores.

## Breve Historia

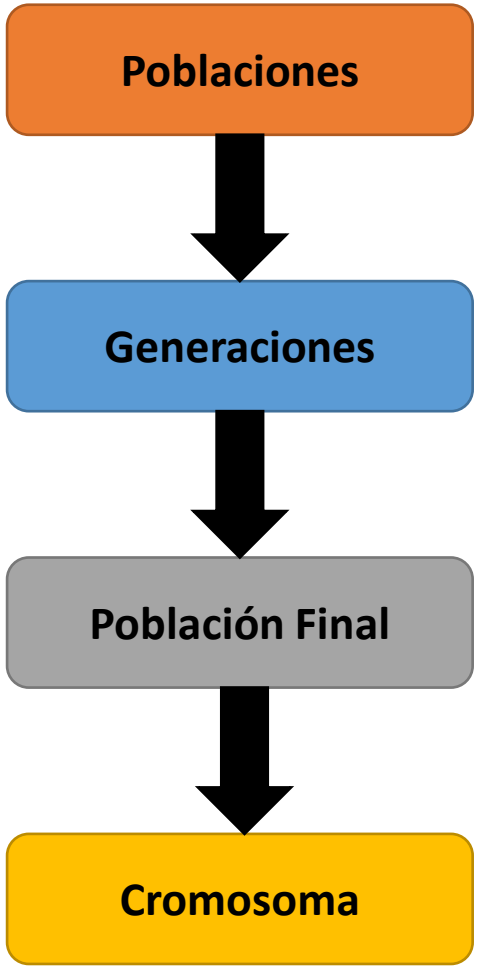
- Los algoritmos genéticos se inspiran en la teoría de la evolución de Charles Darwin (selección natural y supervivencia del más apto).
- Fueron propuestos en los años 60–70 por John Holland en la Universidad de Míchigan.
- Su idea central era utilizar los principios de la biología (mutación, cruce, selección) para resolver problemas complejos de optimización y búsqueda.
- Posteriormente, en los años 80–90, se aplicaron en inteligencia artificial, optimización combinatoria, ingeniería, bioinformática, control y finanzas.

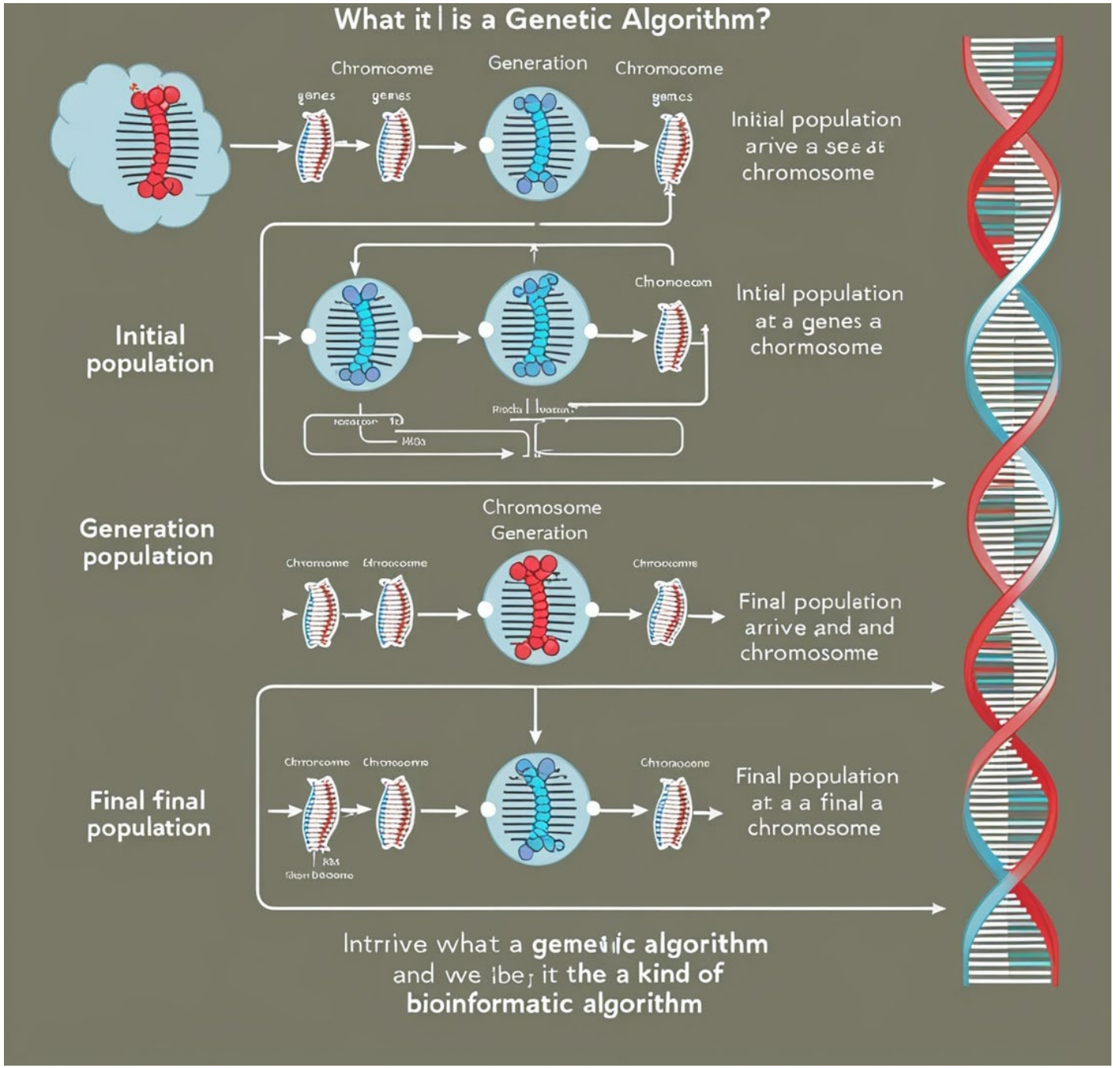
## Ventajas

- No necesita derivadas ni gradientes
- Explora grandes espacios de búsqueda
- Es paralelizable lo que permite la ejecución de múltiples núcleos y máquinas.
- Es flexible, porque se adapta a diferentes problemas de optimización, clasificación, etc.

## Desventajas

- Alto costo computacional.
- No garantizan la mejor solución.
- Parámetros sensibles, dado que el rendimiento depende del tamaño de la población.
- Puede tener una convergencia lenta en algunos problemas.




**Fuente: Leonardo AI**

UNIVERSIDAD SANTO TOMÁS

# GRACIAS