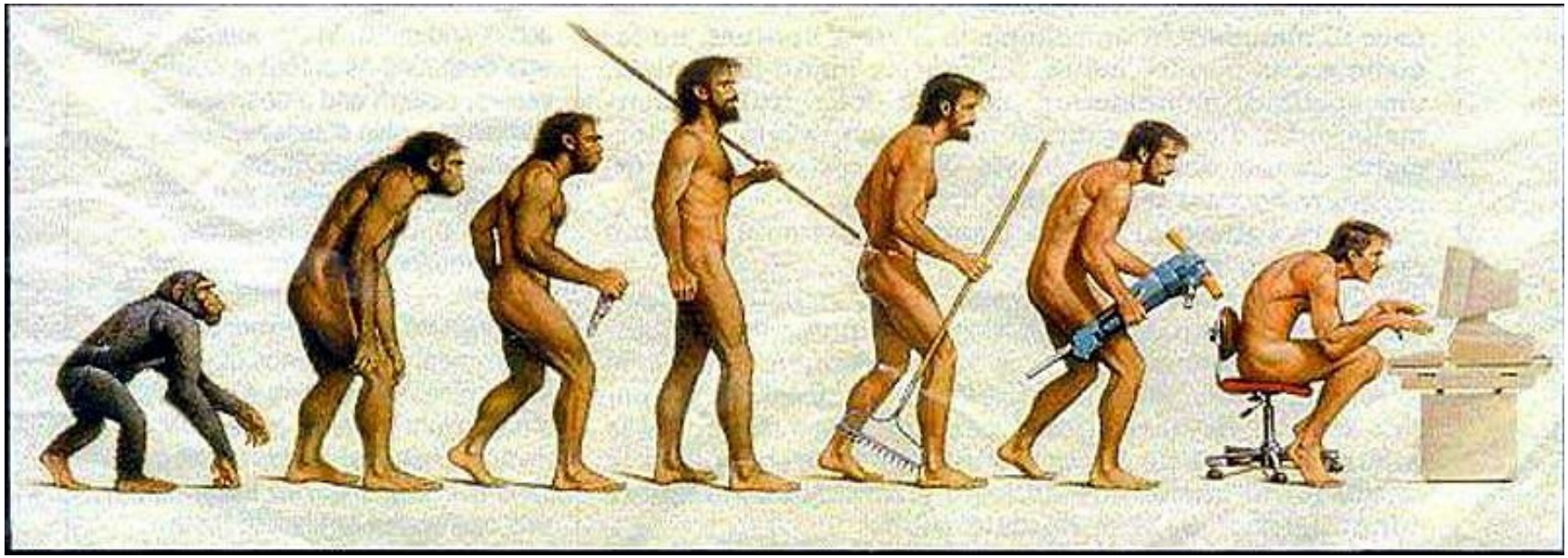
Universidad Autónoma del Estado de México
Facultad de Ingeniería



Introducción al Cómputo Evolutivo

Héctor Alejandro Montes
hamontesv@uaemex.mx
http://scfi.uaemex.mx/hamontes

¿Qué es un Algoritmo Genético?

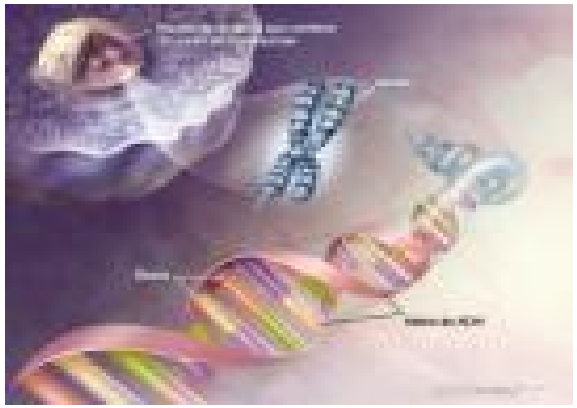
- Los AGs son una familia de algoritmos que se han utilizado para ***optimización***, ***búsqueda*** y ***aprendizaje***
  - inspirados en los procesos de *Evolución Biológica*

Evolución biológica

- En la naturaleza, los procesos evolutivos ocurren cuando se satisfacen las siguientes condiciones:
  - Una entidad o individuo tiene la habilidad de reproducirse.
  - Hay una población de tales individuos que son capaces de reproducirse.
  - Existe variedad entre los individuos que se reproducen.
  - Algunas diferencias en la habilidad para sobrevivir en el entorno están asociadas con esa variedad.

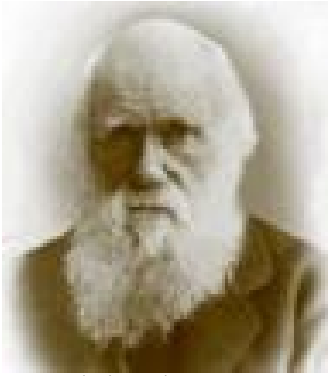
Evolución biológica (2)

- Los mecanismos evolutivos no son del todo conocidos, aunque algunas de sus características son ampliamente aceptadas
- La evolución es un proceso que opera sobre los cromosomas más que sobre las estructuras de la vida que están codificadas en ellos.



Evolución biológica (3)

- La selección natural es el enlace entre los cromosomas y la actuación de sus estructuras decodificadas.
- El proceso de reproducción es el punto en el cual la evolución toma parte, actúa.
  - Darwin, C. (1859). *On the Origin of Species by Means of Natural Selection* or *The Preservations of Favored Races in the Struggle for Life*. London: John Murray.



Evolución artificial

- Modelos inspirados en la Evolución Natural.
- Utilizan "*poblaciones*" que representan soluciones a problemas
- A éstos métodos se les llama *Algoritmos Evolutivos* y se han agrupado en un mismo campo llamado *Cómputo Evolutivo* (CE).

CE: Paradigmas básicos

- Algoritmos Genéticos: poblaciones de representaciones fijas (binarias, permutaciones, reales, etc.), John Holland, 1975 (muere 9/Ago/2015).
- Estrategias de Evolución: *secuencia de números reales* dentro de un rango y variadas sobre una distribución de probabilidad, 1964.
- Programación Evolutiva: evolución de *autómatas de estado finito*, 1960.
- Programación Genética: *expresiones/programas representados como árboles*, 1989.

John Holland
Professor of CS and Psychology at the U. of Michigan

Hans-Paul Schwefel
Universität Dortmund

Ing. Ingo Rechenberg Bionics & Evolutiontechnique Technical University Berlin

Lawrence J. Fogel,
Natural Selection, Inc.

John Koza
Stanford University

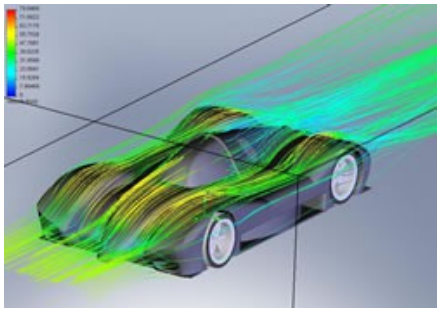
Algunos dominios de aplicación

- Optimización combinatoria (dominios discretos y contínuos
- Modelado e identificación de sistemas
- Planificación y control
- Modelado automotriz
- Vida artificial
- Aprendizaje máquina
- Minería de datos (finanzas, economía, entre otros)
- Procesamiento de Imágenes y Visión Artificial
- Sistemas de Recuperación de Información

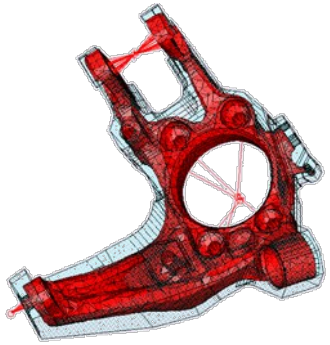
Aplicaciones de uso práctico

- Antena ST5, NASA 2006

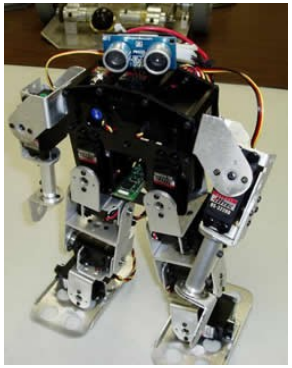
- Modelado automotriz



- Diseño mecánico



- Robótica



¿Qué es un AG?

- Son una clase particular de *Algoritmos Evolutivos*.
- *Se aplican sobre una población representada de forma abstracta como cromosomas, que son la codificación de soluciones candidatas a un problema.*
- La evolución comienza usualmente con una población *aleatoria*.
- En cada *generación*, un método de *selección* elegirá que individuos son aptos, *apareándolos* y *mutándolos* para crear la siguiente *generación*.
- Tienen *ventajas* y también muchas *desventajas*

Estructura de un AG

Procedimiento Algoritmo Genético

Inicio (1)

t = 0;

inicializar P(t);

evaluar P(t);

Mientras (no se cumpla la condición de parada) hacer

Inicio(2)

t = t + 1

seleccionar P(t) desde P(t-1)

recombinar P(t)

mutación P(t)

evaluar P(t)

Final(2)

Final(1)

Usos comunes de un AG

- Búsqueda: Ubicar o localizar ***algo***
- Optimización: Búsqueda de lo ***mejor***
- Aprendizaje: Adquisición de conocimiento a través de ***experiencia***
- En evolución se *buscan* soluciones *mejores* usando conocimientos previos: *experiencia*

Búsqueda Local/Global

- Una búsqueda se realiza sobre un conjunto de soluciones posible llamado *espacio de búsqueda*
- Búsqueda *local*: la siguiente solución esta en el *vecindario* de la solución actual
- Búsqueda *global*: la siguiente solución esta en *cualquier parte* del espacio de búsqueda
- Búsqueda *directa*: No requiere derivar el espacio de búsqueda
- Búsqueda *indirecta*: Usualmente requiere derivar el espacio de búsqueda

Alternativas de búsqueda

- Búsqueda *exhaustiva*: Examina *todas* las posibles soluciones
- Búsqueda *aleatoria:* Genera siguiente solución al *azar* escogida de todo el espacio de búsqueda y repite durante ***n*** iteraciones
- Búsqueda *Hill-climbing*: Genera siguiente solución vecina bajo algún criterio y repite durante ***n*** iteraciones

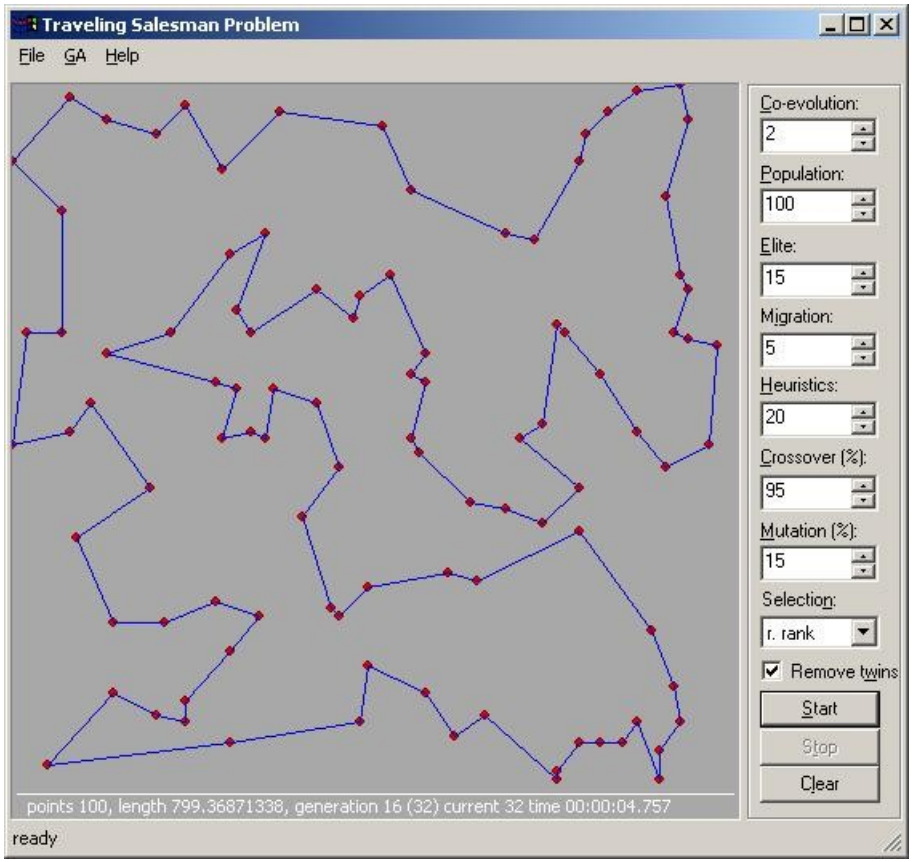
Problema del Agente Viajero
TSP, por sus siglas en inglés

- Problema prototipo en optimización combinatoria
- El TSP es conceptualmente (y engañosamente) simple
- TSP: Dado un conjunto ***n*** de ciudades y una ruta entre cualquier par de ellas, encontrar un ciclo Hamiltoniano que pase por las ***n*** y regrese al inicio
- Cualquier permutacion de las ***n*** ciudades produce una solución
- Número total de soluciones (espacio de búsqueda) ***S = n!***
- ***No se conoce un algoritmo que garantice encontrar el óptimo en tiempos útiles***

Problema del Agente Viajero
TSP, por sus siglas en inglés

- Para 25 ciudades:
  - *15,511,210,043,330,985,984,000,000* posibles soluciones
- Verificar *una ruta* toma algunos cientos de tiempo de CPU
  - Aprox. *1,500,000,000,000,000,000,000,000,000* ciclos
- Los CPU modernos pueden correr a 3 GHZ aprox., es decir, *3,000,000,000* ciclos/seg
  - 25 ciudades tomaría aprox. *500,000,000,000,000,000* seg.
- La edad actual del universo esta estimada en *14 billones de años* o *441,796,964,000,000,000* seg.
- ***… aún nos faltaría otro billión de años***

Problema del Agente Viajero



Konstantin Boukreev, http://www.tsp.gatech.edu/

Búsqueda *exahustiva*

- Búsqueda *exhaustiva*: Examina *todas* las posibles soluciones
- *Brute-force*

```
1: S = first(P) // get first solution from solution set P
2: repeat
    3: R = next(P) // get next solution from P
    4: if Quality(R) > Quality(S) then
        5: S = R
6: until empty(P)
7: return S
```

Búsqueda *aleatoria*

- Búsqueda *aleatoria:* Genera siguiente solución al *azar* escogida *de todo el espacio de búsqueda* y repite durante ***n*** iteraciones
- *Random search*

```
1: S = rand(P) // randomly get first solution from solution set P
2: repeat
    3: R = rand(P) // randomly get next solution from P
    4: if Quality(R) > Quality(S) then
        5: S = R
6: for n iterations
7: return S
```

Búsqueda *Hill-climbing*

- Búsqueda *Hill-climbing*: Genera siguiente solución *vecina* bajo algún criterio y repite durante ***n*** iteraciones
- *Hill-climbing*

```
1: S = rand(P) // get first random solution from solution set P
2: repeat
    3: R = Tweak(Copy(S)) // choose next solution using S
    4: if Quality(R) > Quality(S) then
        5: S = R
6: for n iterations
7: return S
```

Trabajo

- Implemente un programa que genere ***todas*** las posibles soluciones del TSP para un número de ciudades ***n***
- Implemente RS & HC para el TSP con las ***n*** ciudades ordenadas en un ***círculo***
- Grafique sus resultados
- Escriba un breve reporte con el análisis de sus resultados