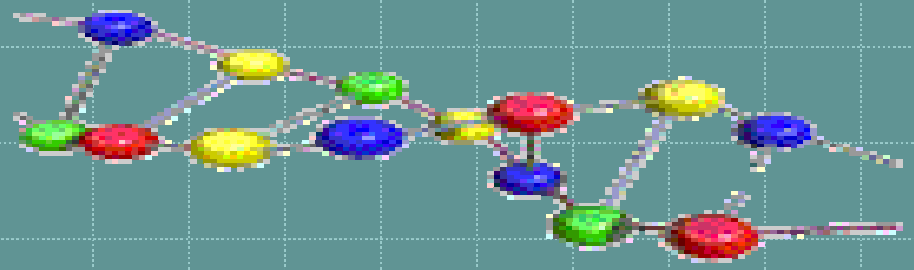
Java y los AG



IEEE ARGENTINA

UP | Universidad de Palermo
Facultad de Ingeniería

Próximas charlas

La IA Evaluación de calidad en Java — (D. López De Luise, M. Agüero) → 09/10/06 (19:00hs)

AG como motor de clasificaciones — (D. López De Luise, F. Goldenstein, G. Barrera) → 07/11/06 (19:00hs)

Análisis de calidad Java con Java — (D. López De Luise, M. Agüero) → 21/11/05 (19:00hs)

informes:

upgrade@palermo.edu ***sec.argentina@ieee.org***

UP

Universidad de Palermo (Depto. Ing. Informática)

IEEE

Institute of Electric and Electronic Engineers (cap. local CIS)

Internetworking the world

Objetivo

- Presentar los AG y Java para su manejo
- Presentar una experiencia concreta en AG con Java

Temario

- Por qué AG?
- Por qué no AG?
- Terminología básica
- Un código Java de ejemplo: TSP

Por qué AG ?

académico

EVOLUCION

infraestrutura global

comercio electrónico

comunicaciones digitales . . .

SPIDERS

VIRUS

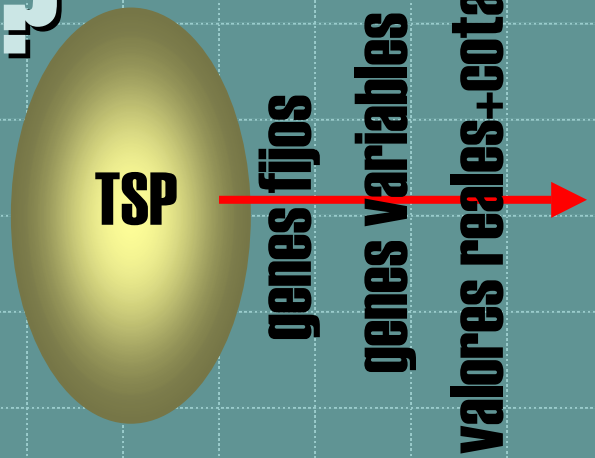
SUPERVIVENCIA

SERVICIO/PRODUCTO COMPETENCIA

Darwin => AG

Por qué no AG ?

- Aplicabilidad limitada



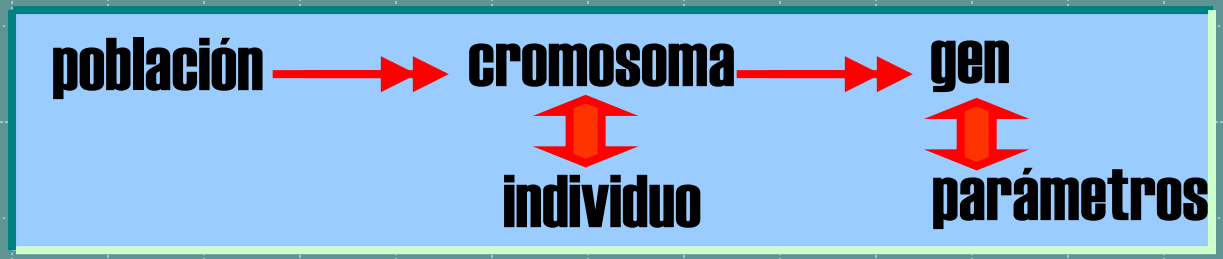
- Sensible a detección de parámetros del problema
- Sensible a representación correcta de parámetros
- Sensible a función de fitness

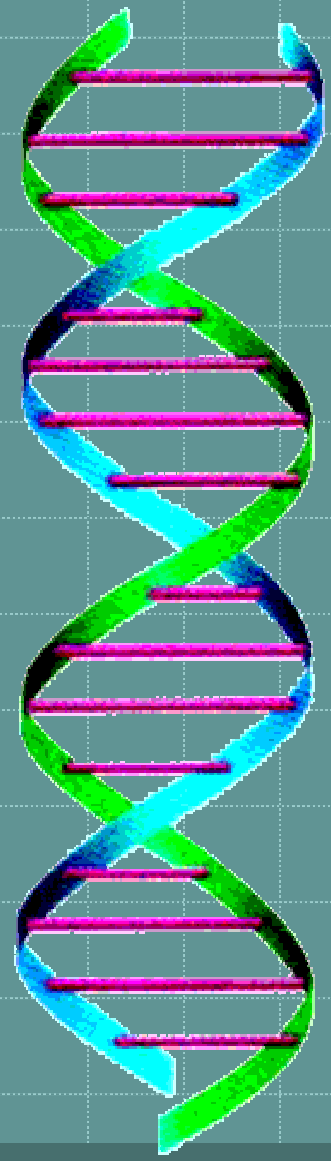
Terminología

CROMOSOMA ═══ INDIVIDUO= PTO. ESPACIO SOLUCIÓN

FN. FITNESS ═══ SELECCIÓN DEL MÁS APTO
(optimización)

GENES ═══ MODELIZACIÓN DE PARÁMETROS





UN CODIGO EJEMPLO

TSP

CROMOSOMA ═══ lista ordenada de ciudades

FN. FITNESS ═══ sumatoria de distancias asociadas dentro del cromosoma

GENES ═══ ciudad y sus coordenadas cartesianas (x, y)

población inicial → tamaño de población inicial

población inicial → tamaño de población normal

```
import java.util.Properties;
public class TSPGene implements IGene {
   private String m_description;
   private double m_x;
   private double m_y;
//
   public TSPGene() {}
   public TSPGene(Properties i_properties, int i_index) {
                    init(i_properties, i_index); }
   public TSPGene(String i_description, double i_x, double i_y)  {
    this.m_description = i_description;
    this.m_x = i_x;   this.m_y = i_y;   }
//
   public void init(Properties i_props, int i_index)
   {    setDescription(i_props.getProperty("gene." + i_index + ".description"))
        setX((Double.valueOf(i_props.getProperty("gene." +
                     i_index + ".x")).doubleValue()));
       setY((Double.valueOf(i_props.getProperty("gene." +
                    i_index +  ".y")).doubleValue()));
1  }
```

Constructores

Carga valores iniciales dde archivo

```
//
public String   getDescription() { return m_description; }
   public void     setDescription(String i_description) {
             m_description = i_description; }
   public double getX() { return m_x; }
   public void      setX(double i_x) { m_x = i_x; }
   public double getY() { return m_y; }
   public void      setY(double i_y) { m_y = i_y; }
```

getter/setter

```
//
   public IGene copy() {
            return new TSPGene(getDescription(),
            getX(), getY()); }
   public boolean equals(IGene i_gene)   {
             return(i_gene.getDescription().equals(m_description));}
   public String toString() {
             return m_description; }
  }
```

métodos de soporte

```
public double getFitness()
  {
    double i_fitness = 0.0;
     for (int i = 0; i < m_genes.size(); i++) {
        TSPGene g1 = (TSPGene) m_genes.at(i);
        TSPGene g2 = null;
        if (i == (m_genes.size() - 1))
                g2 = (TSPGene) m_genes.at(0);
        else
                g2 = (TSPGene) m_genes.at(i+1);
        i_fitness += calcDistance(g1.getX(), g1.getY(),   g2.getX(), g2.getY());
     }
     m_fitness = i_fitness; return m_fitness;}

  private static double calcDistance(double x1, double y1,  double x2,
                                                              double y2)
  {
     double deltaXSquared = (x2 - x1) * (x2 - x1);
     double deltaYSquared = (y2 - y1) * (y2 - y1);
     return Math.pow(deltaXSquared + deltaYSquared , 0.5);   }
```

función de fitness

distancia cartesiana entre (x1, y1) y (x2, y2)

```
// SELECCION: Retorna 2 objetos IChromosome selecionados
   protected IChromosome[ ] selectParents(int i_tournamentSize) {
       IChromosome[ ] parents = new IChromosome[2];
       Array gladiators = new Array(i_tournamentSize);

      int startIndex = GeneticAlgorithmUtils.getRandomInt(0,
       m_population.size() - i_tournamentSize);

for (int i = startIndex; i < (startIndex + i_tournamentSize); i++)
          gladiators.put(i - startIndex, m_population.at(i));
          sort(gladiators);

          parents[0] = (IChromosome) gladiators.at(0);
          parents[1] = (IChromosome) gladiators.at(1);
         return parents;
 }
```

selecciona un grupo de individuos al azar

considera los i_tournamentSize cromosomas

ordena los cromosomas por fitness

retorna los 2 mejores en parents

```
// Cruza de cromosomas
public IChromosome[ ] crossover(IChromosome i_chromosome,
                                double i_crossoverRate)  {
  int size = this.getSize();
  IChromosome mom = this.copy();
  IChromosome dad = i_chromosome.copy();
  IChromosome[ ] children = new IChromosome[2];

double random=GeneticAlgorithmUtils.getRandomDouble(1.0);
  if (random < i_crossoverRate) {
     int crossoverPoint=GeneticAlgorithmUtils.getRandomInt(1,size-1);
     IGene[ ] child1begin=mom.getGenes(0, crossoverPoint-1);
     IGene[ ] child1end=getEndGenes(child1begin,dad.getGenes(0,size-1));
     IGene[ ] child2begin = dad.getGenes(0, crossoverPoint - 1);
     IGene[ ] child2end = getEndGenes(child2begin, mom.getGenes(0,size -1));
     children[0] = new TSPChromosome(child1begin, child1end);
     children[1] = new TSPChromosome(child2begin, child2end);
     return children;  }

  children[0] = mom;
  children[1] = dad;
  return children;}
```

cruza el cromosoma *this* con otro

sin cruza los hijos son los padres

preserva unicidad genes en el cromosoma

retorna 2 cromosomas hijos

```
// Mutacion
  public void mutate(double i_mutationRate) {
    double random = GeneticAlgorithmUtils.getRandomDouble(1.0);
    if (random < i_mutationRate) {
      int size = this.getSize();

      int r1 = GeneticAlgorithmUtils.getRandomInt(0, size - 1);
      int r2 = GeneticAlgorithmUtils.getRandomInt(0, size - 1);

      IGene g1 = (IGene) this.getGene(r1);
      IGene g2 = (IGene) this.getGene(r2);
      m_genes.put(r2, g1);
      m_genes.put(r1, g2);
    }
  }
```

muta un i_mutationRate% de veces

seleciona 2 genes al azar y los intercambia

```
// insercion
protected void insert(IChromosome[ ] i_children, int i_elitism)
  {
     sort(m_population);

     for (int i = i_elitism; i < m_population.size(); i++)
        m_population.put(i, i_children[i - i_elitism]);

  }
```

preserva los i_elitism mejores cromosomas de la generación anterior

inserta cromosomas hijos


fin