



Sistemas complejos. Algoritmos evolutivos y bioinspirados

Taller de herramientas para sistemas complejos: algoritmos evolutivos

Pedro A. Castillo Valdivieso
Baeza, 13 de marzo de 2006



Contenidos del taller

- Introducción**
- Estructura de un algoritmo evolutivo**
- Material para este taller**
- ¿Qué es Algorithm::Evolutionary?**
- Uso de individuos binarios**
- Uso de un operador de mutación**
- Uso de un operador de cruce**
- Ejemplo. Algoritmo genético simple**
- Ejemplo. Algoritmo evolutivo**



Introducción

Los algoritmos evolutivos están inspirados en la evolución natural

Son métodos para resolución de problemas que aplican los métodos de la evolución biológica:

- selección basada en población
- reproducción sexual
- mutación

Un tutorial sobre computación evolutiva:

<http://geneura.ugr.es/~jmerelo/ie/ags.htm>



Introducción

Los AE son métodos de optimización

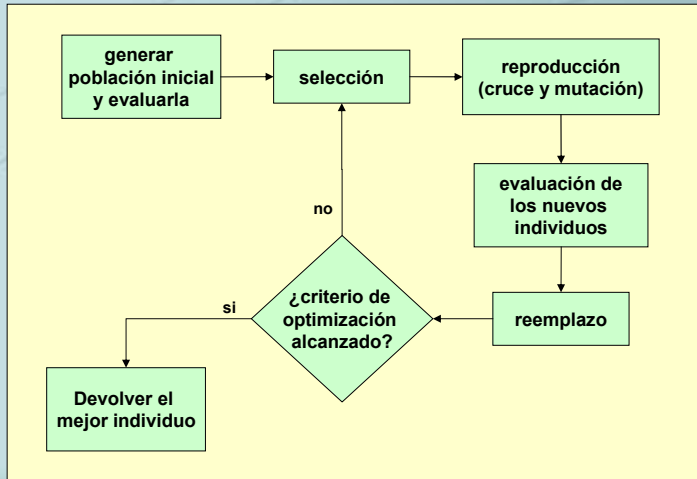
- Parametrizar el problema en una serie de variables
- Codificar las variables en un cromosoma (solución)
- Aplicar operadores de variación a estas soluciones

Las soluciones compiten, y sólo las mejores (las que resuelven mejor el problema) sobreviven

El AE se usará normalmente para optimizar sólo una función (frente a los algoritmos multiobjetivo)



Bucle general de un AE



Elementos del AE: individuos

Las soluciones candidatas codifican las variables del problema.

Hay que elegir la representación más adecuada al problema.

- Cadenas binarias
- Vectores de números reales
- Redes neuronales artificiales
- Grafos
- Árboles de decisión
- Etc, etc.



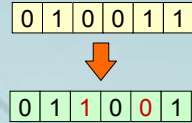
Elementos del AE: operadores genéticos

Generan nuevos individuos a partir de los que ya están en la población

MUTACIÓN:

Modifica sólo unos genes del cromosoma.

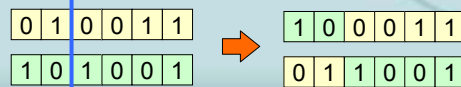
Contribuye a la diversidad de la población



CRUCE:

Intercambio de material genético entre cromosomas.

Tomar dos padres, cortarlos por N puntos, e intercambiar genes



Elementos del AE: evaluación (fitness)

Decodificar el cromosoma para extraer las variables del problema

Evaluar el problema con esas variables.

Obtener una puntuación (fitness).

El valor de fitness determina qué individuos se van a reproducir y cuáles se eliminarán



Material para este taller

- Documentación
- Intérprete de lenguaje Perl
- Paquete de la librería Algorithm::Evolutionary
- Ejemplos utilizados

<http://geneura.ugr.es/~pedro/cursos/baeza/ae/>



¿Qué es Algorithm::Evolutionary?

Librería de programación para hacer computación evolutiva

Diseñada y desarrollada por Juan Julián Merelo

<http://opear.sourceforge.net>

- Abstrae las características de los paradigmas evolutivos, permitiendo resolver problemas de forma rápida y casi sin escribir código
- Es fácil programar cualquier tipo de algoritmo evolutivo
- Están disponibles todas las representaciones para los individuos y todos los operadores genéticos



Instalación y configuración de *Perl*

Perl es un lenguaje interpretado

En cuanto a sintáxis se parece al lenguaje C

En Unix / Linux se encuentra instalado por defecto.

Para instalar una versión para Windows, podemos descargar:

<http://www.activestate.com>

Para ampliar conocimientos sobre el lenguaje (características, estructuras de datos y de control, etc) podemos descargar un tutorial de:

<http://geneura.ugr.es/~pedro/cursos/perl/>



Creación y ejecución de programas *Perl*

El aspecto de un programa muy sencillo es el siguiente:

```
#!c:/perl/bin/perl.exe

print " hola mundo ! " ;
```

```

c:\cmd
C:\perl\TALLER\1\ejemplos>
C:\perl\TALLER\1\ejemplos>
C:\perl\TALLER\1\ejemplos>
C:\perl\TALLER\1\ejemplos>
C:\perl\TALLER\1\ejemplos>
C:\perl\TALLER\1\ejemplos>
C:\perl\TALLER\1\ejemplos>
C:\perl\TALLER\1\ejemplos>
C:\perl\TALLER\1\ejemplos>
C:\perl\TALLER\1\ejemplos>
C:\perl\TALLER\1\ejemplos>type miprograma.pl
#!c:/perl/bin/perl.exe
print " hola mundo ! " ;
C:\perl\TALLER\1\ejemplos>
C:\perl\TALLER\1\ejemplos>perl.exe miprograma.pl
hola mundo !

```



Instalación de la librería

C:\> perl.exe -MCPAN -e shell

```

dnmake.exe - all
21.
main::createAndTest() called too early to check prototype at t\individuals.t line
e 28.
Name "main::op" used only once: possible typo at t\individuals.t line 29.
t\individuals.....ok
t\NoChangeTerm.....ok
t\ops.....main::createAndTest() called too early to check prototype
at t\ops.t line 31.
t\ops.....ok
t\run.....Can't locate XML/LibXML.pm in @INC (@INC contains: ..\..
..\..\..\..\..\C:\cpan\build\Algorithm-Evolutionary-0.53\lib\lib C:\cpan\build\
Algorithm-Evolutionary-0.53\lib\arch C:/perl/lib C:/perl/site/lib) at t\run.t l
ine 6.
BEGIN failed--compilation aborted at t\run.t line 6.
t\run.....dubious
Test returned status 2 (wstat 512, 0x200)
t\validate.....Can't locate XML/LibXML.pm in @INC (@INC contains: C:\cpan
\build\Algorithm-Evolutionary-0.53\lib\lib C:\cpan\build\Algorithm-Evolution
ary-0.53\lib\arch C:/perl/lib C:/perl/site/lib) at t\validate.t line 2.
BEGIN failed--compilation aborted at t\validate.t line 2.
t\validate.....dubious
Test returned status 2 (wstat 512, 0x200)
Failed Test Stat Wstat Total Fail Failed List of Failed
-----
t\run.t          2    512    ??    ??    x    ??
t\validate.t     2    512    ??    ??    x    ??
t subtest skipped.
Failed 2/9 test scripts, 77.78% okay. 0/102 subtests failed, 100.00% okay.
dnmake.exe: Error code 2, while making 'test_dynamic'
dnmake test -- NOT OK
Running make install
Appending installation info to C:\Perl\lib\perllocal.pod
dnmake install -- OK

cpan>
cpan>

```

Ha terminado de instalar la librería. Todo está listo



¿Cómo usar la librería?

Usar un editor de textos para crear el programa Perl

A partir de un ejemplo que funcione, programar nuestra función de fitness, elegir el tipo de dato (individuo) que se ajuste al problema, y usar los operadores genéticos adaptados a esa representación

Ejecutar en una ventana de comandos

C:\> perl.exe miprograma.pl



Definición de individuos binarios

Un algoritmo genético clásico usa individuos binarios.

```
#!c:/perl/bin/perl.exe
```

```
use Algoritmo
```

```
my $indiv =
```

```
Algoritmo
```

```
print $:
```

```
print $:
```

```

C:\perl\taller_baeza06\ejemplos>
C:\perl\taller_baeza06\ejemplos>type 0_bits.pl
#!c:\perl\bin\perl.exe
use strict;
use warnings;

use Algorithm::Evolutionary::Individual::BitString;

my $indi3 = Algorithm::Evolutionary::Individual::BitString->new (<10>);

print "\nI=".$indi3->atom(< 7 >)."\n";
print "\nI=".$indi3->asString()."\n";

C:\perl\taller_baeza06\ejemplos>
C:\perl\taller_baeza06\ejemplos>
C:\perl\taller_baeza06\ejemplos>perl 0_bits.pl

[0]
[[1110110000 -> ]

```



Uso de un operador de mutación

Debemos incluir el módulo "Mutation".

```
#!c:/perl/bin.
```

```
use Algoritmo
```

```
use Algoritmo
```

```
my $mutar = A.
```

```
my $indiv =
```

```
Algoritmo
```

```
my $mutado = :
```

```
print $indiv.
```

```
print $mutado.
```

```

C:\perl\taller_baeza06\ejemplos>type 1_mutacion.pl
#!usr/bin/perl
use warnings;
use strict;

use Algorithm::Evolutionary::Individual::BitString;
use Algorithm::Evolutionary::Op::Mutation;

my $m = Algorithm::Evolutionary::Op::Mutation->new(<0.1 >);
my $indiv = Algorithm::Evolutionary::Individual::BitString->new (<8>);
my $mutado = $m->apply(<$indiv >);

print "\n\n";
print "\nI = " . $indiv->asString() . "\n";
print "\n\n";
print "\nM = " . $mutado->asString() . "\n";
print "\n\n";

C:\perl\taller_baeza06\ejemplos>perl.exe 1_mutacion.pl

I= 10001110 ->
M= 10000110 ->

```



Uso de un operador de cruce

Debemos incluir el módulo "Crossover".

```

#!c:/perl/bin/perl.exe
use Algorithm::Evolutionary;
use Algorithm::Evolutionary::Op::Crossover;

my $cruce = Algorithm::Evolutionary::Op::Crossover->new( 1 );
my $i1 = Algorithm::Evolutionary::Individual::BitString->new( 8 );
my $i2 = Algorithm::Evolutionary::Individual::BitString->new( 8 );
my $hijo1 = $cruce->apply( $i1, $i2 );
my $hijo2 = $cruce->apply( $i2, $i1 );

print "\n\n";
print "\nI1= " . $i1->asString() . "\n";
print "\nI2= " . $i2->asString() . "\n";

print "\n\n";
print "\nH1= " . $hijo1->asString() . "\n";
print "\nH2= " . $hijo2->asString() . "\n";

print "\n\n";

C:\perl\TALLER\1\ejemplos>perl.exe 1_cruce.pl

I1= 0100 ->
I2= 0010 ->

H1= 0110 ->
H2= 0000 ->

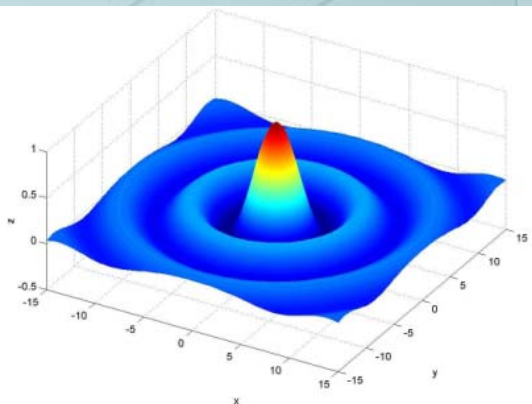
```



Resolver un problema con un algoritmo genético

$$f(x,y) = \frac{\sin(\sqrt{x^2 + y^2})}{\sqrt{x^2 + y^2}}$$

máximo en (0,0)
con $f(0,0) = 1$





Resolver el problema con un algoritmo genético

Debemos seleccionar el tipo de individuo, los operadores y programar la función de fitness:

- Cadenas binarias
Individual::BitString
- Operadores genéticos adaptados a la codificación binaria
Op::Mutation
Op::Crossover
- La función de evaluación es la función a optimizar



Elementos del algoritmo genético

Los módulos necesarios para este ejemplo

```
use Algorithm::Evolutionary::Individual::BitString;  
use Algorithm::Evolutionary::Op::Easy;  
use Algorithm::Evolutionary::Op::Mutation;  
use Algorithm::Evolutionary::Op::Crossover;
```

La población de soluciones

```
my @pop;  
my $i;  
for ( $i=0 ; $i < $popSize ; $i++ ) {  
    my $indi = Algorithm::Evolutionary::  
        Individual::BitString->new( $numBits ) ;  
    $pop[$i] = $indi ;  
}
```



Elementos del algoritmo genético

La función de evaluación (fitness)

```
my $funcionMarea = sub {
    my $chrom = shift;
    my $str = $chrom->Chrom();
    my $fitness = 1;
    my $l2=length($str)/2;
    my $x=eval("0b".substr ($str, 0, $l2));
    my $y=eval("0b".substr ($str, $l2));
    my $max=(2 ** ($l2) )-1;
    $x=$x/$max;          # los pasamos al rango [0,1]
    $y=$y/$max;
    my $sqrt=sqrt( ($x*$x) + ($y*$y) );
    if($sqrt !=0 ){ $fitness = sin( $sqrt ) / $sqrt; }
    return $fitness;
};
```



Elementos del algoritmo genético

El bucle del algoritmo

```
my $generation = Algorithm::Evolutionary::Op::Easy->new(
    $funcionMarea , 0.2 , [$m, $c] );
do {
    $generation->apply( \@pop );
    $numGens -- ;
} while( $numGens > 0 );
```

Una vez terminado, mostramos la mejor solución

```
print "La mejor solución encontrada es: ";
print $pop[0]->asString() ;
```



Resolver el problema con un algoritmo evolutivo

Resolver la función marea utilizando codificación real

$$f(x,y) = \frac{\sin(\sqrt{x^2 + y^2})}{\sqrt{x^2 + y^2}}$$

Vectores de números reales

Individual::Vector

Operadores genéticos adaptados a la codificación real

Op::GaussianMutation

Op::VectorCrossover

El resto del algoritmo queda igual que el estudiado antes



Elementos del AE...

Los módulos necesarios para este ejemplo

```
use Algorithm::Evolutionary::Individual::Vector;
use Algorithm::Evolutionary::Op::Easy;
use Algorithm::Evolutionary::Op::GaussianMutation;
use Algorithm::Evolutionary::Op::VectorCrossover;
```

La función de fitness (marea)

```
my $funcionMarea = sub {
    my $indi = shift;
    my ( $x, $y ) = @{$indi->[_array]};
    my $sqrt = sqrt( $x*$x+$y*$y);
    if( !$sqrt ){ return 1; }
    return sin( $sqrt )/$sqrt;
};
```



Elementos del AE...

Los operadores genéticos

```
my $m =  
    Algorithm::Evolutionary::Op::GaussianMutation->new(0, 0.1);  
my $c =  
    Algorithm::Evolutionary::Op::VectorCrossover->new(2);
```

Al término del algoritmo, mostrar la solución

```
my ( $x, $y ) = @{$pop[0]->{_array}};  
  
print "El mejor es:\n\t ";  
print $pop[0]->asString() ;  
print "\n\t x=$x \n\t y=$y \n\t Fitness: ";  
print $pop[0]->Fitness() ;
```



Resumen

- Fácil de instalar y de usar
- Lenguaje similar a C++
- Minimizar la cantidad de código a escribir
- Posibilidad de ampliación, añadiendo nuevos tipos de algoritmos, representaciones y operadores
- Código abierto