Implementación del AGS

El SGA (simple genetic algorithm)(Goldberg, 1989) es un algoritmo genético que incorpora los siguientes métodos y criterios:

- Criterio de codificación: Específico de cada problema. Debe hacer corresponder a cada punto del dominio del problema un elemento del espacio de búsqueda: cadenas binarias.
- Criterio de tratamiento de los individuos no factibles: No hay. Se considera que la codificación se hace de tal manera que todas las cadenas posibles representan a individuos factibles.
- Criterio de inicialización: La población inicial está formada por cadenas binarias generadas al azar.

Implementación del AGS

- Funciones de evaluación y aptitud: La función de aptitud coincide con la de evaluación. La función de evaluación viene dada a través de una función objetivo.
- Operadores genéticos: Cruce monopunto y mutación bit a bit sobre los individuos codificados.
- Criterio de selección: Por sorteo.
- Criterio de reemplazo: Inmediato.
- Criterio de parada: Fijando el número máximo de iteraciones.
- Parámetros de funcionamiento: Discrecionales. Una elección posible es

$$TamPob = 30 \quad MaxIter = 50 \quad p_{cru} = 40\,\% \quad p_{mut} = 1{,}0\,\%$$

Implementación del AGS

- El cruce monopunto genera dos descendientes a partir de dos progenitores cortándolos en una posición elegida al azar e intercambiando los respectivos segmentos.
- Dados dos progenitores codificados como $\mathbf{v} = \langle b_1 \dots b_\ell \rangle$ y $\mathbf{w} = \langle c_1 \dots c_\ell \rangle$ con $b_j, c_j \in 0, 1$ $(\forall j = 1, \dots, \ell)$ se genera un número aleatorio $pos \leftarrow \mathrm{AleaEnt}[1, \ell - 1]$ y se procede así:

$$\langle b_1, \dots, b_{pos-1} | b_{pos}, \dots, b_\ell \rangle \qquad \langle b_1, \dots, b_{pos-1} | c_{pos}, \dots, c_\ell \rangle$$

$$\Longrightarrow$$

$$\langle c_1, \dots, c_{pos-1} | c_{pos}, \dots, c_\ell \rangle \qquad \langle c_1, \dots, c_{pos-1} | b_{pos}, \dots, b_\ell \rangle$$

Implementación del AGS

- Cada aplicación de la mutación bit a bit conmuta un bit de entre todos los $(n \times \ell)$ de la población.
- La determinación de las parejas a cruzar y a mutar se realiza de acuerdo con las probabilidad de aplicación de los operadores genéticos.
- Para el cruce:
  - Para cada individuo se genera un número aleatorio $r_i \leftarrow \mathrm{Alea}[0, 1)$.
  - Se comparan los r_i con la probabilidad de cruce, p_{cru}, que es un parámetro del método.
  - Son seleccionados para cruzarse todos los individuos $\mathbf{v}_i$ para los que

$$r_i < p_{cru}$$

Implementación del AGS

- Para la mutación se realiza el mismo proceso con cada uno de los $n \times \ell$ bits de la población y usando la probabilidad de mutación p_{mut}.
- El emparejamiento de los individuos a cruzar se hace al azar: las etapas anteriores al cruce han introducido el suficiente desorden en la población como para que un emparejamiento sucesivo se pueda considerar aleatorio.
- Cuando el número de individuos a cruzar es impar, el último individuo quedará desemparejado y se elimina o se empareja con otro elegido al azar.

Implementación

- Consideremos una posible implementación en pseudocódigo del algoritmo genético simple.
- Consideraremos operadores de reproducción, cruce y mutación aplicados a la optimización de una función simple de una variable codificada como un entero binario sin signo.

El Algoritmo

```
funcion Alg_Gen (Parametros_alg)
{
    TPoblacion pob;    // población
    entero tam_pob;   // tamaño población
    entero lcrom;       // tamaño individuo
    entero num_max_gen; // número máximo de generaciones
    entero pos_mejor; // posición del mejor cromosoma
    real prob_cruce;    // probabilidad de cruce
    real prob_mut;      // probabilidad de mutación
    real x_min, x_max;      // extremos del intervalo considerado
    real tol;      // tolerancia de la representación
```

El Algoritmo

```
    obtener_parametros(tam_pob, lcrom, x_min, x_max,
                       pob_cruce, prob_mut, tol, num_max_gen);
    pob = poblacion_inicial(tam_pob, lcrom, ...);
    evaluacion(pob, tam_pob, pos_mejor);
    // bucle de evolución
    para cada generacion desde 0 hasta num_max_gen hacer{
        seleccion(pob, tam_pob);
        reproduccion(pob, tam_pob, prob_cruce, ...);
        mutacion(pob, tam_pob, prob_mut, ...);
        evaluacion(pob, tam_pob, pos_mejor);
    }
}
```

Estructuras de Datos

- Representación del genotipo:

  tipo TGenes : **vector** de **booleano**;

- Cada individuo incluye información sobre su genotipo y fenotipo (el valor real que representa) y otros datos necesarios para su evaluación, como su valor de aptitud, su puntuación relativa respecto a los restantes individuos, y su puntuación acumulada, usada en los sorteos.

Estructuras de Datos

- **tipo** TIndividuo = **registro**{
  TGenes genes; // cadena de bits (genotipo)
  real x; // fenotipo
  real aptitud; // funcion de evaluacion
  real puntuacion; // puntuacion relativa : aptitud / suma
  real punt_acu; // puntuacion acumulada para sorteos
  }TIndividuo;
- Representación de la población:
  tipo TPoblacion: **vector** de TIndividuo;

Cálculo de $lcrom$

- Buscamos una codificación: representación del dominio del problema mediante enteros binarios sin signo.
- La elección más sencilla consiste en discretizar el intervalo $[x_{min}, x_{max}]$ en una cantidad de puntos 2^{lcrom} tal que la distancia entre puntos consecutivos sea menor que la tolerancia especificada,

$$\frac{x_{max} - x_{min}}{2^{lcrom} - 1} < TOL$$

- Cada punto del espacio de búsqueda queda representado mediante un entero binario de longitud $lcrom$, comenzando por `0...0` que representa a x_{min} y terminando en `1...1` que representa a x_{max}.

Cálculo de $lcrom$

- La longitud de los individuos se calcula a partir de la tolerancia TOL:

$$lcrom = \left\lceil \log_2 \left(1 + \frac{x_{max} - x_{min}}{TOL} \right) \right\rceil$$

- Recíprocamente, se puede calcular el punto x que corresponde a un individuo $\mathbf{v}$ mediante la siguiente fórmula de decodificación:

$$x(\mathbf{v}) = x_{min} + \text{bin2dec}(\mathbf{v}) \cdot \frac{x_{max} - x_{min}}{2^{lcrom} - 1}$$

Generación de la Población Inicial

```
TPoblacion poblacion_inicial(entero tam_pob, entero lcrom)
{
    TPoblacion pob;
    TIndividuo indiv;
    entero i;
    para cada i desde 0 hasta tam_pob {
        indiv = genera_indiv(lcrom);
        pob.insertar(indiv);
    }
    devolver pob;
}
```

La creación de cada individuo consiste en generar su genotipo como una cadena aleatoria de ceros y unos, y en calcular su aptitud.

Generación de la Población Inicial

```
TIndividuo genera_indiv(entero lcrom, ...)
{
    entero i;
    TIndividuo indiv;
    entero gen;
    indiv.genes.iniciar();
    para cada i desde 0 hasta lcrom hacer {
        si (alea() < 0.5 ) entonces
            gen = 0;
        eoc
            gen = 1;
        indiv.genes.insertar(gen);
    }
    indiv.aptitud = adaptacion(indiv, ...);

    devolver indiv; }
```

Adaptación de los Individuos

La función de aptitud consiste en decodificar el genotipo del individuo para identificar al punto que representa, y calcular el valor de la función objetivo en ese punto.

```
funcion adaptacion(TIndividuo individuo, ...)
{
    real x; // fenotipo
    real f; // valor de la funcion a optimizar
    x = decod(individuo.genes, ...);
    f = f(x);
    devolver f;
}
```

Adaptación de los Individuos

La decodificación de un individuo se realiza calculando el valor decimal correspondiente al genotipo del individuo, y dividiendo por el número de puntos en el que se ha discretizado el intervalo o espacio de búsqueda considerado.

```
funcion decod(TGenes genes, ...)
{
    real x;
    x = x_min + (x_max - x_min) *
            bin_dec(genes, lcrom) / (pow(2,lcrom) - 1);
    devolver x;
}
```

Adaptación de los Individuos

```
funcion bin_dec(TGenes genes, entero lcrom)
{
    entero i, d=0, pot=1;
    para cada i desde 0 hasta lcrom hacer{
        d = d + pot*genes[i];
        pot = pot * 2;
    }
    devolver d;
}
```

Evaluación de la Población

Después de cada generación se revisan los contadores de aptitud relativa, y puntuación acumulada de los individuos de la población. Así mismo, se calcula la posición del mejor individuo.

```
funcion evaluacion(TPoblacion& pob, entero tam_pob,
                                    entero& pos_mejor)
{
    real punt_acu = 0; // puntuación acumulada
    real aptitud_mejor = 0; // mejor aptitud
    real sumaptitud = 0; // suma de la aptitud
    entero i;
```

Evaluación de la Población

```
    para cada i desde 0 hasta tam_pob hacer {
        sumaptitud = sumaptitud + pob[i].aptitud;
        si (pob[i].aptitud > aptitud_mejor){
            pos_mejor = i;
            aptitud_mejor = pob[i].aptitud;
        }
    }
    para cada i desde 0 hasta tam_pob hacer {
        pob[i].puntuacion = pob[i].aptitud / sumaptitud;
        pob[i].punt_acu = pob[i].puntuacion + punt_acu;
        punt_acu = punt_acu + pob[i].puntuacion;
    }
}
```

Selección y Reproducción

- Suponemos la existencia de procedimientos de generación de número aleatorios:
  - *alea*
    Devuelve un número real pseudoaleatorio entre cero y uno (una variable aleatoria uniforme en el intervalo real [0,1]).
  - *alea_ent*
    Devuelve un valor entero entre los límites inferior y superior especificados.

Selección y Reproducción

- Los operadores de selección, cruce y mutación se combinan para producir una nueva generación.
- En un AGS, el proceso comienza con la selección de los individuos que sobreviven.
- La implementación de la reproducción consiste en la selección de los individuos a reproducirse entre los de la población resultante, y en la aplicación del operador de cruce a cada una de las parejas.
- En todos los casos de realiza la mutación bit a bit mediante la función booleana mutation.

Selección

La función de selección escoge, por el método de sorteo, un número de supervivientes igual al tamaño de la población. La función modifica la población que pasa a estar formada únicamente por ejemplares de los individuos supervivientes.

```
funcion seleccion(TPoblacion& pob, entero tam_pob)
{
    entero sel_super[tam_pob]; // seleccionados
                               // para sobrevivir
    real prob; // probabilidad de seleccion
    entero pos_super; // posición del superviviente
    TPoblacion pob_aux; // población auxiliar
    entero i;
```

Selección

```
para cada i desde 0 hasta tam_pob hacer {
    prob = alea();
    pos_super = 0;
    mientras ((prob > pob[pos_super].punt_acu) y
                    (pos_super < tam_pob))
       pos_super++;
   sel_super[i] = pos_super;
}
// se genera la poblacion intermedia
para cada i desde 0 hasta tam_pob hacer {
   pob_aux.insertar(pob[sel_super[i]]);
}
pob ⟵ pob_aux;
}
```

Reproducción

```
funcion reproduccion(TPoblacion& pob, entero tam_pob,
                                    real prob_cruce, ...)
{
    real sel_cruce[tam_pob]; // seleccionados
                                        // para reproducirse
    entero num_sel_cruce=0; // contador seleccionados
    real prob;
    entero punto_cruce;
    TIndividuo hijo1, hijo2;
    entero i;
```

Reproducción

```
// se eligen los individuos a cruzar
para cada i desde 0 hasta tam_pob {
    // se generan tam_pob números aleatorios a_i en [0 1)
    prob = alea();
    // se eligen los individuos de las posiciones i
    // si prob < prob_cruce
    si (prob < prob_cruce){
        sel_cruce[num_sel_cruce] = i;
        num_sel_cruce++;
    }
}
// el numero de seleccionados se hace par
si ((num_sel_cruce mod 2) == 1)
    num_sel_cruce–;
```

Reproducción

```
    // se cruzan los individuos elegidos en un punto al azar
    punto_cruce = alea_ent(0,lcrom);
    para cada i desde 0 hasta num_sel_cruce avanzando 2{
        cruce(pob[sel_cruce[i]], pob[sel_cruce[i+1]]
                                 hijo1, hijo2, punto_cruce, ...);
        // los nuevos individuos sustituyen a sus progenitores
        pob[sel_cruce[i]] = hijo1;
        pob[sel_cruce[i+1]] = hijo2;
    }
}
```

Cruce

El operador de cruce toma dos padres y genera dos cadenas hijas. Recibe la probabilidad de cruce. La función calcula la aptitud de los nuevos individuos.

```
funcion cruce(TIndividuo padre1, TIndividuo padre2,
              TIndividuo& hijo1, TIndividuo& hijo2,
              entero punto_cruce, ...)
{     entero i;
    hijo1.genes.iniciar();
    hijo2.genes.iniciar();
```

Cruce

```
    // primera parte del intercambio: 1 a 1 y 2 a 2
    para cada i desde 0 hasta punto_cruce hacer{
        hijo1.genes.insertar(padre1.genes[i]);
        hijo2.genes.insertar(padre2.genes[i]);
    }
    // segunda parte: 1 a 2 y 2 a 1
    para cada i desde punto_cruce hasta lcrom; hacer{
        hijo1.genes.insertar(padre2.genes[i]);
        hijo2.genes.insertar(padre1.genes[i]);
    }
   // se evalúan
    hijo1.aptitud = adaptacion(hijo1, . . . );
    hijo2.aptitud = adaptacion(hijo2, . . . );
}
```

Mutación

El operador de mutación considera la posible mutación de cada gen del genotipo con la probabilidad indicada. La función revisa la aptitud del individuo en caso de que se produzca alguna mutación.

```
mutacion mutacion(TPoblacion& pob, entero tam_pob,
                                   real prob_mut, ...)
{
    booleano mutado;
    entero i,j;
    real prob;
```

Mutación

```
para cada i desde 0 hasta tam_pob hacer{
    mutado = falso;
    para cada j desde 0 hasta lcrom hacer{
        // se genera un numero aleatorio en [0 1)
        prob = alea();
        // mutan los genes con prob < prob_mut
        si (prob < prob_mut){
            pob[i].genes[j] = not( pob[i].genes[j]);
            mutado = cierto;
        }
    si (mutado)
        pob[i].aptitud = adaptacion(pob[i], ...);
    }
}
}
```