Algoritmos Genéticos

Grupo A:
Adrián Racero Serrano
Santiago Ponce Arrocha
Carlos Velasco Hurtado
Juan Manuel Cardeñosa Borrego

```java
public static Cromosoma crossover(Cromosoma c1, Cromosoma c2) {
    Cromosoma hijo = new Cromosoma(); //hijo fruto del cruce de c1 y c2
    Random r = new Random();

    int puntoCruce = r.nextInt(SIZE - 1) + 1; //seleccionamos de forma aleatoria el punto de cruce (entre 1 y 8)

    //Copiamos la primera parte de c1
    for (int i = 0; i < puntoCruce; i++) {
        for (Tupla t : c1.cromosoma[i]) {
            hijo.addValue(i, t.pos, t.value);
        }
    }

    //Copiamos la segunda parte de c2
    for (int i = puntoCruce; i < SIZE; i++) {
        for (Tupla t : c2.cromosoma[i]) {
            hijo.addValue(i, t.pos, t.value);
        }
    }

    return hijo;
}
```

Crossover

Objetivo del mutar: intercambiar los valores dentro del cromosoma (de forma aleatoria, no las posiciones).
Si intercambio posición: al mostrar cromosoma, los valores quedan desordenados.

```java
public static Cromosoma mutate(Cromosoma c) {
    Cromosoma mutant = new Cromosoma(c); //cromosoma fruto de la mutacion de c
    Random r = new Random();


    int whereToMutate = r.nextInt(SIZE); //en que parte del cromosoma se va a mutar (entre 0 y 8)
    int portionSize = c.cromosoma[whereToMutate].size(); //longitud de la porcion del cromosoma

    if (portionSize > 1) {
        int posicion1, posicion2;
        do {
            posicion1 = r.nextInt(portionSize);
            posicion2 = r.nextInt(portionSize);
        } while (posicion1 == posicion2);

        Tupla t1 = mutant.cromosoma[whereToMutate].get(posicion1); //variable auxiliar para el intercambio
        Tupla t2 = mutant.cromosoma[whereToMutate].get(posicion2); //variable auxiliar para el intercambio
        mutant.cromosoma[whereToMutate].set(posicion1, new Tupla(t1.pos, t2.value)); //intercambiamos los valores de posicion1 con posicion2 (queremos
        mutant.cromosoma[whereToMutate].set(posicion2, new Tupla(t2.pos, t1.value)); //intercambiamos los valores de posicion2 con posicion1
    }

    return mutant;
}
```

Necesito como mínimo 2 valores para mutar

Mutate

Fitness

```java
public int fitness(Cromosoma c){
    int valoresUnicos = 0; //Contador de valores unicos

    //Contamos valores unicos en las columnas
    for (int i = 0; i < SIDE; i++) {
        ArrayList<Integer> numeros = new ArrayList<>();
        for (int j = 0; j < SIDE; j++) {
            int n = tablero[j][i];
            if(n == 0){ //Si n vale 0, miramos al cromosoma
                n = c.getValue(j,i);
            }
            if(!numeros.contains(n)){ //si no hemos guardado ese valor previamente, es unico
                numeros.add(n);
                valoresUnicos++;
            }
        }
    }
    //Contamos para los cuadrados
    //Iterar sobre los cuadrados
    for (int i = 0; i < SIDE; i+=3) {
        for (int j = 0; j < SIDE; j+=3) {

            //Iteramos dentro de cada cuadrado 3x3
            ArrayList<Integer> numeros = new ArrayList<>();
            for (int k = i; k < i + 3; k++) {
                for (int l = j; l < j + 3; l++) {
                    int n = tablero[k][l];
                    if(n == 0){ //Si n vale 0, miramos al cromosoma
                        n = c.getValue(k,l);
                    }
                    if(!numeros.contains(n)){ //si no hemos guardado ese valor previamente, es unico
                        numeros.add(n);
                        valoresUnicos++;
                    }
                }
            }
        }
    }
    //Devolvemos los valores unicos que hemos contado
    return valoresUnicos;
}
```

Solve

```java
public void solve(){
    ArrayList<Cromosoma> population = new ArrayList<>(poblacionInicial);
    ArrayList<Cromosoma> population2 = new ArrayList<>();
    Cromosoma bestIndividual = bestFitness(population);
    int timer = 0;
    System.out.println(fitnessMedio(population) + " " + fitness(bestIndividual));

    while(fitness(bestIndividual) != MAXFITNESS && timer<100000) {
        for (int i = 0; i < population.size(); i++) {
            Cromosoma primero, segundo;
            Random r = new Random();
            int size;
            Set<Integer> set;
            int probabilidadMutar; //numero entre el 0 y el 100
            if(fitness(bestIndividual) >= 156){
                size = 8;
                probabilidadMutar = 80;
            } else {
                size = 4;
                probabilidadMutar = 30;
            }

            set = new HashSet<>();
            while(set.size() < size){
                set.add(r.nextInt(population.size()));
            }
            ArrayList<Cromosoma> competicion = new ArrayList<>();
                for (Integer n : set) {
                    competicion.add(population.get(n));
                }
                primero = bestFitness(competicion);
                competicion.remove(primero);
                segundo = bestFitness(competicion);

            Cromosoma hijo = Cromosoma.crossover(primero, segundo);

            if(r.nextInt(bound:100) < probabilidadMutar){
                hijo = Cromosoma.mutate(hijo);
            }
            population2.add(hijo);
        }
        population = new ArrayList<>(population2);
        population2 = new ArrayList<>();
        bestIndividual = bestFitness(population);
        timer++;
        System.out.println(fitnessMedio(population) + " " + fitness(bestIndividual));
    }
    if(fitness(bestIndividual) == MAXFITNESS){
        resuelto = true;
    }
    reconstructSudoku(bestIndividual);
}
```

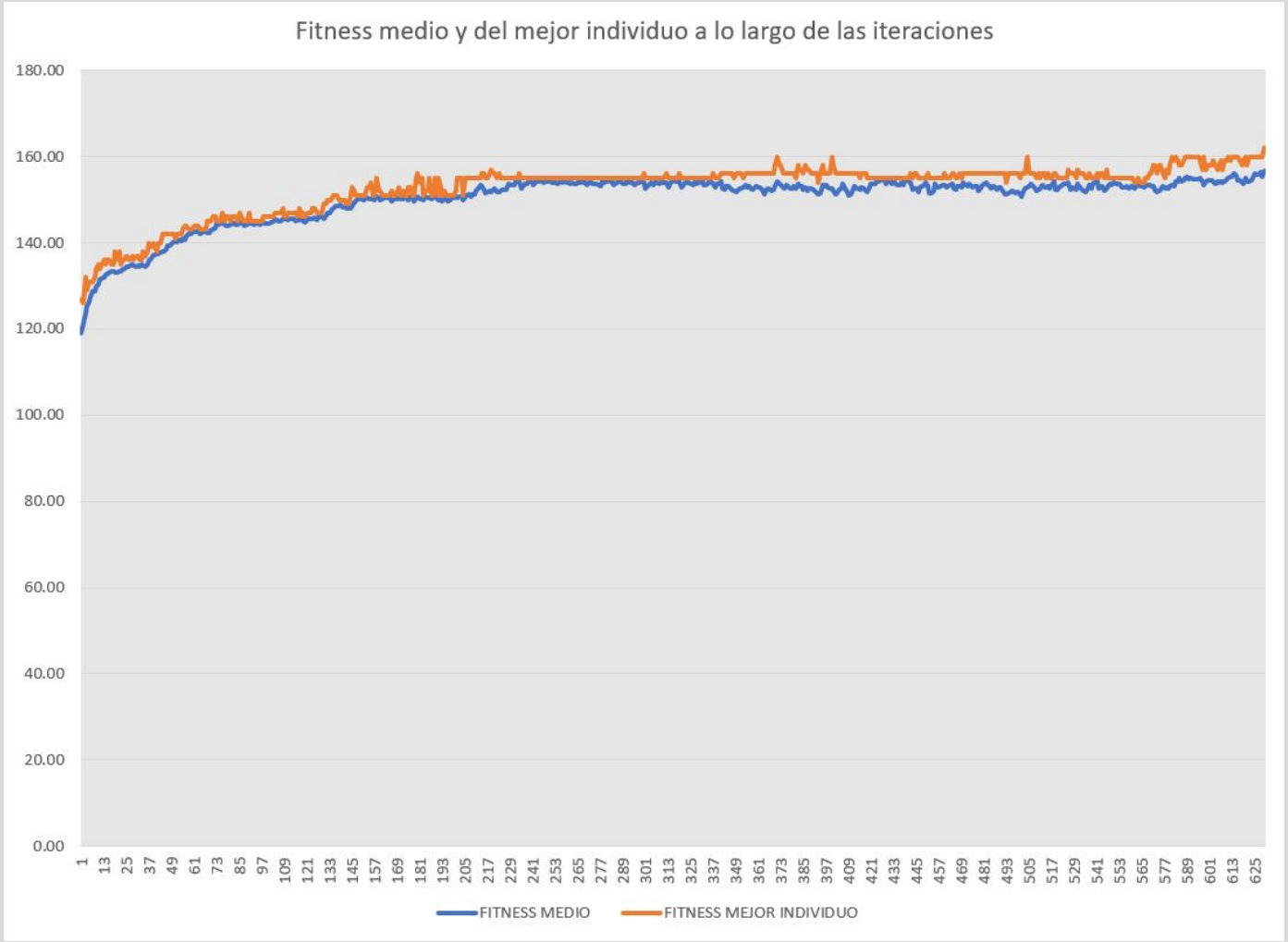
Problemas surgidos

El algoritmo genético resultaba muy lento antes de implementar la estrategia de competición y de cambiar el parámetro para la probabilidad de mutar. Sigue resultando lento para sudokus complejos, pero los sencillos los resuelve rápidamente.

Conclusión

Es una práctica muy interesante, la cual nos ha permitido entender cómo funcionan más a fondo los algoritmos genéticos, tanto sus ventajas (adaptables a muchos problemas, en algunos casos son eficientes, ...) como sus desventajas (puede no encontrar la solución óptima, para problemas complejos puede tardar mucho, ...).

PARTE VOLUNTARIA


Fitness medio y del mejor individuo a lo largo de las iteraciones
180.00
160.00
140.00
120.00
100.00
80.00
60.00
40.00
20.00
0.00
1
13
25
37
49
61
73
85
97
109
121
133
145
157
169
181
193
205
217
229
241
253
265
277
289
301
313
325
337
349
361
373
385
397
409
421
433
445
457
469
481
493
505
517
529
541
553
565
577
589
601
613
625
FITNESS MEDIO
FITNESS MEJOR INDIVIDUO