Algoritmo Genético

El algoritmo genético utilizado corresponde al algoritmo genético generacional. Para la aplicación del algoritmo genético se siguen los siguientes pasos:

1. Crear la población inicial
2. Evaluar la población inicial
3. Seleccionar los elementos de la población que van a ser ocupados en la reproducción
4. Cruzar los elementos de la población para generar la decendencia
5. Mutar la decendencia.
6. Repetir 3 4 5 hasta que se alcance el número de iteraciones indicados.

La población inicial se genera usando la función de la clase de java Random llamada nextInt. La función nextInt utilizada recibe como parámetro un limite superior, pero necesitamos que el valor este entre un limite inferior y superior indicado por nosotros. Debido a lo anterior, creamos una funciono extra que hace uso de nextInt para generar un valor entre dos límites. Esta nueva función usa la siguiente ecuación:

limite\_inferior + Random.nextInt(limite\_superior - limite\_inferior);

Por ejemplo, si tenemos los siguientes valores:

limite\_inferior = 10;

limite\_superior = 20;

entonces al aplicar la función tendremos lo siguiente

10 + Random.nextInt(10)

Y como la función de java genera valores entre 0 y el límite superior ingresado (sin incluir a este) entonces los valores de Random.nextInt estarán entre el rango [0 , 9] resultando en que el rango de los valores generados por nuestra nueva función estará entre [10, 19].

# Evaluación

Para evaluar la población primero necesitamos los valores que posee el archivo gama, ya que este contiene los diámetros y los costos. Para esto, se necesita una clase donde guardar los valores de una línea del archivo, esta clase la llamamos Gama. Al momento de leer el archivo gama se guarda cada línea en un ArrayList de objetos Gama, cada posición dentro de este ArrayList es una línea del archivo gama. Cuando el algoritmo genético manda a evalúa la solución ve cada elemento individual de esta que corresponde a un índice dentro del ArrayList de Gamas. Puesto que, los elementos de la solución están entre el rango [1, cantidad de elementos dentro de la gama] y el ArrayList está entre el rango [0, Cantidad de elementos dentro de la gama], al usar los elementos de la solución con el ArrayList hay que restarles 1. Por ejemplo, si tenemos el siguiente caso:

Solucion : (1,2, 3, 4, 1, 2, 3)

Y la gama posee

|  |  |  |
| --- | --- | --- |
| Índice | Diámetro | Costo |
| 0 | 10 | 1000 |
| 1 | 20 | 2000 |
| 2 | 30 | 3000 |
| 3 | 40 | 4000 |

Para obtener el diámetro asociado a solución[3], cuyo índice es 4. Hay que hacer gama.get(4-1), lo cual nos daría el objeto Gama ubicado en la 3 posición de la tabla anterior, el cual contiene un diámetro de 40 y un costo de 4000.

Con este índice se obtiene el costo y el diámetro y luego hay que calcular el valor del objetivo usando la siguiente ecuación:

Donde n es la cantidad de elementos dentro de la solución, largo\_i es el largo del elemento en la posición i dentro del archivo inp y costo\_i es el costo asociado al índice en la solución. Luego, hay que evaluar la factibilidad, para esto se envía la solución la cual, ya tiene el valor objetivo, a un objeto evaluador llamado MonoObjetiveEvaluator.

MonoObjetiveEvaluator posee una función llamada evaluate que recibe la solución enviada por la función evaluate del objeto Problem y el ArrayList de las gamas. Dentro de este evaluate del MonoObjetiveEvaluator se llama a la librería de epanet usando la EpanetAPI, con el fin de realizar la simulación y comprobar si es factible la solución. Para esto, usando la función de EpanetAPI.setlinkvalue, se cambian los diámetros que fueron cargados por defecto desde el archivo inp y se realiza la simulación. Cuando se realiza la simulación se obtiene el valor de la presión asociada a cada nodo usando la función getnodelinkvalue de EpanetAPI. Y si la presión dada es menor que la presión mínima ingresada en el problema se aumenta un contador (numberOfViolationConstrains) y se calcula un grado de infactibilidad que consiste en lo siguiente:

Donde n es el numero de nodos de la red, obtenidos a través de la función getcount de EpanetAPI; y presión\_i es la presión del i-esimo nodo obtenida a través de la función getnodelinkvalue.

El numero de violaciones a las restricciones y el grado de infactibilidad se registran en la solucion usando las clases NumberOfViolatedConstraints y OverallConstraintViolation, respectivamente.

# Selección

Para la selección se utiliza el operador de selección explicado en la tesis del profesor Daniel more, llamado por nosotros como UniformSelection. Este operador lo que hace es recibir una lista de soluciones y ordenarlas teniendo en cuenta el valor asociado al grado de infactibilidad y el valor del objetivo, lo cual deja las mejores soluciones al principio y las peores al final. Luego, a la mejor solución se le calcula una probabilidad que corresponderá a la máxima probabilidad permitida para ese conjunto de soluciones. Mientras que a la peor solución se le calcula la probabilidad mínima. El cálculo de la probabilidad máxima y mínima se realiza de la siguiente manera:

Donde B es una constante entre 1,5 y 2 y Nc es el tamaño de la población.

Luego a los restantes elementos se le asigna una probabilidad basada en su posición en la lista y usando la siguiente ecuación:

Una vez asociadas la probabilidad a cada elemento de la lista se procede a crear una nueva lista en donde las soluciones cuyo producto pi\*Nc es mayor a 1.5 se agrega dos veces. Si pi\*Nc está entre 0.5 y 1.5 se agrega una vez. Si el producto esta bajo ese valor entonces no se agrega. Esta nueva lista creada corresponde a la lista a usar para el cruzamiento y la mutación.

# Reproducción

La reproducción consiste en el uso de 2 operadores, el de cruzamiento (crossover) y el de mutación (mutatión).

## Crossover

El crossover realizado consiste en el de cruzamiento basado en un solo punto de cruce. Este crossover recibe 2 soluciones. Luego, se genera un valor random usando la función random que permite ingresar un límite inferior y superior explicada anteriormente. Con esta función random se obtiene un índice a partir del cual se intercambiarán los valores de las soluciones. Como se muestra en la siguiente imagen:

[Ingrese imagen aqui]

Una vez realizado este cambio se devuelven las 2 soluciones resultantes.

## Mutación

Una vez que se tienen las soluciones entregadas por el operador Crossover, sobre cada una de ellas se aplica el operador de mutación. El operador utiliza una probabilidad de mutación ingresada al momento de crearlo. Cuando se esta ejecutando el operador se recorre cada elemento de la solución y al mismo tiempo se genera un valor aleatorio entre 0 y 1 usando el método getNextDouble de la clase Random de java. Si este valor generado es menor que el de la probabilidad de mutación, entonces se cambia el valor del elemento por un valor aleatorio entre el limite inferior y limite superior permitido.

[Imagen de mutacion aqui]

Todo lo mencionado anteriormente se repite hasta que se alcanza una cantidad de iteraciones máxima indicada al momento de crear el algoritmo genético.