El Poeta,
Proyecto III análisis de algoritmos

Nelson Rojas Obando y Jose Luís Rodriguez
Estudiantes de
Ingeniería en Computación
Instituto Tecnológico de Costa Rica
Carnés: 2016062840, 2016093725

***Abstract*—A continuación se presentan algoritmos para la solución de generación automática de texto. Para el desarrollo de este proyecto se utilizaron algoritmos genéticos con distancias como "Manhattan", "Chebyshev" y además una distancia creada originalmente.**

Este proyecto busca generar, a partir de un poema de entrada, nuevos poemas que tengan relación con el ingresado pero hayan sido generados a partir de combinaciones de otros poemas.

I. Introduction

Los algoritmos genéticos permiten la generación de poblaciones nuevas a partir de una población base. Un algoritmo genético es un método de búsqueda que imita la teoría de la evolución biológica de Darwin para la resolución de problemas. Para ello, se parte de una población inicial de la cual se seleccionan los individuos más capacitados para luego reproducirlos y mutarlos y finalmente obtener la siguiente generación de individuos que estarán más adaptados que la anterior generación. Estos algoritmos presentan un comportamiento no determinístico, esto es, que no siempre se llegará a la misma solución. Se busca, por medio de funciones de adaptabilidad, obtener un resultado cercano al esperado. Entre menor sea la distancia obtenida de la función de adaptabilidad, mayor será la similitud entre el resultado generado y el resultado objetivo, que para efectos del presente proyecto corresponden a poemas.

November 14, 2017

II. Distancias utilizadas

A. Distancia Manhattan

```
private int[] metahistogram;
private int[] generatedhistogram;
private List<int> indicesPrometedores =
    new List<int>();
private List<int> cantPrometedores = new
    List<int>();
private List<int> indicesNoPrometedores =
     new List<int>();

public ManhattanDistance(int[]
    metahistogram, int[]
    generatedhistogram){
     this.metahistogram = metahistogram;
     this.generatedhistogram =
         generatedhistogram;
     calcular_Distancia();
}

private void calcular_Distancia(){
  int tamanio = metahistogram.Length;
  int distancia = 0;
  for (int i = 0; i < tamanio; i++){
    int d1 = metahistogram[i];
    int d2 = generatedhistogram[i];
    int suma = Math.Abs(d1 - d2);
    distancia += suma;
    if (suma == 0)
      if (d1 != 0){
        indicesPrometedores.Add(i);
        if (d2 >= 2)
          cantPrometedores.Add(d2 - 1);
        else
          cantPrometedores.Add(d2);
      }
    else
      if (d2 != 0 && d1 != 0){
        indicesPrometedores.Add(i);
        if (d2 >= 2)
          cantPrometedores.Add(d2 - 1);
        else
          cantPrometedores.Add(d2);
      }
    }
  }
}
```

Análisis
Este algoritmo tiene como entrada el tamaño del vector del histograma meta "n", únicamente compara y realiza operaciones matemáticas de orden constante.
Por lo que el

$$O(n) = n$$

Tomando como barómetro la comparación que se encuentra dentro del for.

B. Chebyshev

```
private int[] metahistogram;
private int[] generatedhistogram;
private List<int> indicesPrometedores =
    new List<int>();
private List<int> cantPrometedores = new
    List<int>();
private List<int> indicesNoPrometedores =
     new List<int>();
private List<int> distancias = new List<
    int>();
private List<int> ordenList = new List<
    int>();
private List<int> indexList = new List<
    int>();

public ChebyshevDistance(int[]
    metahistogram, int[]
    generatedhistogram){
     this.metahistogram = metahistogram;
     this.generatedhistogram =
         generatedhistogram;
     calcular_Distancia();
}

private void calcular_Distancia(){
  int tamanio = metahistogram.Length;
  int distancia = 0;
  for (int i = 0; i < tamanio; i++){
    int d1 = metahistogram[i];
    int d2 = generatedhistogram[i];
    int suma = Math.Abs(d1 - d2);
    if (suma > distancia)
      distancia = suma;
    else
      if (d1 != 0){
        indicesPrometedores.Add(i);
        cantPrometedores.Add(d2);
      }
  }
}
```

Análisis

De igual modo que el algoritmo anterior, este algoritmo tiene como entrada el tamaño del vector del histograma meta ”n”, además únicamente realiza operaciones matemáticas básicas. Por lo que el

$$O(n) = n$$

Tomando como barómetro la comparación que se encuentra dentro del for.

C. Distancia propia

```
private int[] metahistogram;
private int[] generatedhistogram;
private List<int> indicesPrometedores =
    new List<int>();
private List<int> cantPrometedores = new
    List<int>();
private List<int> indicesNoPrometedores =
     new List<int>();
private List<int> distancias = new List<
    int>();
private List<int> ordenList = new List<
    int>();
private List<int> indexList = new List<
    int>();

public OwnDistance(int[] metahistogram,
    int[] generatedhistogram){
     this.metahistogram = metahistogram;
     this.generatedhistogram =
         generatedhistogram;
     calcular_Distancia();
}

private void calcular_Distancia(){
  int tamanio = metahistogram.Length;
  double distancia = 0;
  for (int i = 0; i < tamanio; i++){
    int d1 = metahistogram[i];
    int d2 = generatedhistogram[i];
    double suma;
    suma = Math.Pow(((d1 + d1) / (d2 + d1
        + 1)) - ((d2 + d1) / (d1 + d2 +
        1)), 2);
    distancia += suma;
    if (suma < 2)
      if (d1 != 0){
        indicesPrometedores.Add(i);
        if (d2 == 2)
          cantPrometedores.Add(d2 - 1);
        else
          cantPrometedores.Add(d2);
      }
  }
}
```

Análisis

De igual modo que los algoritmos anteriores, este algoritmo tiene como entrada el tamaño del vector del histograma meta ”n”, y además únicamente realiza operaciones matemáticas básicas dentro del for, que poseen orden constante.
Por lo que el

$$O(n) = n$$

Tomando como barómetro la comparación que se encuentra dentro del for.

III. Algoritmo genético

La entrada de este algoritmo corresponde a el tamaño del vector generado (poema generado) y el tamaño del poema meta.

```
private string poem = "";
```

```
private int metaSize;
private Dictionary<int, string>
    dicDictionary;
private Random random;

//n * C
public string gen(Dictionary<int, string>
     dictionary, int tamanioMeta){
  //C
  dicDictionary = dictionary;
  metaSize = tamanioMeta;
  random = new Random();
  int c = 0;
  int tamanio = dictionary.Count;
  //n
  while (c < tamanioMeta){
    //C
    int ranIndex = random.Next(tamanio);
    string[] cant = dictionary[ranIndex].
        Split(default(string[]),
        StringSplitOptions.
        RemoveEmptyEntries);
    poem += dictionary[ranIndex];
    c += cant.Length;
  }
  return poem;
}

//m * C
public static List<T> DesordenarLista<T>(
    List<T> input){
    //C
    List<T> arrDes = new List<T>();
    //m
    while (input.Count > 0){
      //C
      int val = random.Next(0, input.
          Count - 1);
      arrDes.Add(input[val]);
      input.RemoveAt(val);
    }
    return arrDes;
  }
//Barometro
//m^2 * c + m * c + m * c
public string new_Generation(List<int>
    prometedores, List<int>
    cantPrometedores, List<string>
    listaTotal){
  //C
  int tamanio = prometedores.Count;
  poem = "";
  List<string> newGenList = new List<
      string>();
  //m^2
  for (int i = 0; i < tamanio; i++){
    int cantP = cantPrometedores[i];
    //m
    for (int j = 0; j < cantP; j++){
      newGenList.Add(listaTotal[
          prometedores[i]]);
    }
  }
  int c = newGenList.Count;
  tamanio = dicDictionary.Count;
  //m
  while (c < metaSize){
    int ranIndex = random.Next(tamanio);
    string[] cant = dicDictionary[
        ranIndex].Split(default(string[]),
        StringSplitOptions.
        RemoveEmptyEntries);
    if (newGenList.Contains(dicDictionary
        [ranIndex]) == false)
    {
      newGenList.Add(dicDictionary[
          ranIndex]);
      c += cant.Length;
    }
  }
  //m * c
  newGenList = DesordenarLista(newGenList
      );
  //m
  foreach (string s in newGenList){
    poem += s + "␣";
  }
  poem.Replace("␣", "");
  return poem;
}
```

Análisis

De este algoritmo se toma como barometro

$$O(n) = m^2 * c + m * c + m * c$$

y de aquí se concluye

$$O(n) = m^2$$

A. N-Grams

Para los N-Grams se toma como entrada el poema meta y el repositorio de poemas con que se cuenta. Esto genera un conjunto mayor con la unión de ambas entradas, de este modo la entrada está definida como el tamaño de la unión de las palabras que componen el poema meta y las palabras que componen el repositorio de poemas.

Además, se generan N-Grams de diferente tamaño. Esto repercute en el número de iteraciones necesarias para obtener las combinaciones en cada caso.

Por todo esto el tamaño de la entrada para el algoritmo que genera N-Grams está definida como

$$N = [(\text{tamaño poema + tamaño repositorio})$$

x Numero de N-Grams]

O visto de otra forma

$$N = [(\text{TP} + \text{TR}) \text{ x NG}]$$

```
public void Create_Ngram(int ngram){
  int index = -1;
  //result = union (poema y repositorio)
  //TP + TR
  foreach (var word in result){
    //c
    index++;
    int cont = 0;
    string ngramTemp = "";
    //NG
    while (cont < ngram){
      //c
      //Barometro (TP + TR) x  NG x C
      if (index + cont < result.Length){
        ngramTemp += result[index + cont]
            + " ";
        cont++;
      }else{
        cont++;
      }
    }
    indexDictionary.Add(indexT, ngramTemp
        );
    indexT++;
  }
  indexDictionary.Distinct();
}
```

Análisis
Así, el orden del algoritmo generador de N-Grams es:

$$O(n) = (TP + TR)xNG$$

B. Histogramas

La función generadora de histogramas toma dos lista, una contiene las palabras que componen el poema meta, mientras que la otra, contiene las palabras del repositorio de poemas. De esto, se forma una lista general la cual es la unión de ambas listas anteriormente mencionadas.

Así, la entrada del algoritmo está conformada por el tamaño de ambas listas. Por lo que tenemos
N = tamaño del poema meta
M = tamaño del repositorio de poemas

```
private List<string> generatedList = new
    List<string>();
private List<string> metaPoemList = new
    List<string>();
private List<string> totalList = new List
    <string>();
private int[] histogramGenerated;
private int[] histogramMeta;
public Histogram(Dictionary<int, string>
    diccionario){
  metaPoemList = metaPoem.Split(default(
      string[]), StringSplitOptions.
      RemoveEmptyEntries).ToList();
  generatedList = generated.Split(default
      (string[]), StringSplitOptions.
      RemoveEmptyEntries).ToList();
  List<string> ls = diccionario.Values.
      ToList();
  totalList = metaPoemList.Union(
      generatedList).Union(ls).Distinct().
      ToList();
  histogramGenerated = new int[totalList.
      Count];
  histogramMeta = new int[totalList.Count
      ];
  histogramGenerated.Initialize();
  histogramMeta.Initialize();
  CreateHistogram();
}

private void CreateHistogram(){
  //N
  foreach (string s in metaPoemList){
    int index = totalList.IndexOf(s);
    histogramMeta[index]++;
  }
  //M
  //Barometro M * c
  foreach (string s in generatedList){
    int index = totalList.IndexOf(s);
    histogramGenerated[index]++;
  }
}
```

Análisis
Del algoritmo se tiene:

$$O(n) = N + M$$

Puesto que se espera que el tamaño del repositorio de poemas sea mucho mayor que el tamaño del poema meta se asume que:

$$O(n) = M$$

IV. Experimentos

Para este proyecto programado se analizaron los datos obtenidos a raíz de varios experimentos realizados, el primero de ellos fue una comparación entre las tres distancias que se implementaron en el programa, los resultados obtenidos reflejan su comportamiento. Ver figura 1.

Como podemos observar, los valores que generan las distancias son muy diferentes entre sí. Sin embargo, se aprecia cierta similitud entre el comportamiento de la distancia de Manhattan y la distancia propia; esto se debe a que ambas

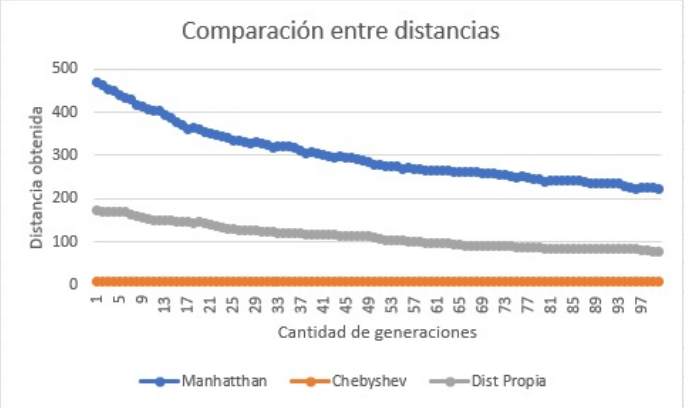

Fig. 1. Comparación entre las tres funciones de distancia aplicadas

utilizan una sumatoria de las distancias calculadas de cada uno de sus genes, mientras que la distancia de Chebishev únicamente contempla la mayor distancia que se obtiene de la comparación de cada uno de los genes, por esta razón gráficamente Chebishev parece no variar. Cabe rescatar que la comparación anterior se realizó con n-grams de hasta 3, también se realizó el mismo experimento con diferentes tamaños de n-grams para observar si existe algún cambio, sin embargo ni con n-grams mas pequeños ni con n-grams más mayores se logró notar una diferencia considerable. A pesar de esto, el gráfico nos demuestra cómo el emplear un algoritmo genético acerca a los poemas generados cada vez más al poema objetivo. A pesar de esto, los algoritmos genéticos tienen su límite tal y como se refleja en la figura 2.

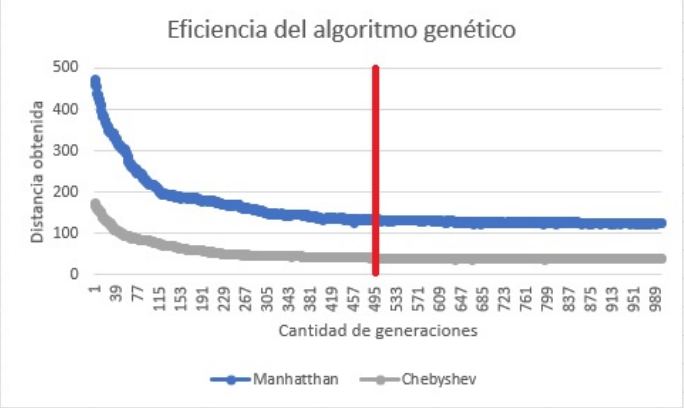

Fig. 2. Punto en el cual el algoritmo no puede mejorar más

Como se puede observar en el gráfico anterior, el algoritmo es eficiente hasta alrededor de la generación 495, más allá de esta el algoritmo parece no tener mayor efecto de mejora en la distancia para el poema generado.

V. Conclusión

Los algoritmos genéticos permiten la generación automática de atributos para encontrar solución a un problema. Estos algoritmos deben de ser guiados por medio de funciones de adaptabilidad para fijar los parámetros que deben de seguir las nuevas generaciones.

Es a partir de las funciones de adaptabilidad que el algoritmo puede determinar la generación de los nuevos elementos de una especie. Además se puede concluir que para obtener mejores resultados, es mejor establecer la creación de un gran número de generaciones, pues entre más generaciones sean evaluadas se obtendrá un mejor resultado en el desempeño de los individuos.