

**SISTEMAS INTELIGENTES.**

**PRÁCTICA DE BÚSQUEDA 3**

**Algoritmos Genéticos**

**1.** En esta práctica vamos a trabajar con la librería de algoritmos genéticos disponible en http://jgap.sourceforge.net/.

En el campus virtual está disponible un problema que es resuelto mediante algoritmos genéticos, *GeneticosHolaMundo.zip* . Este problema es conocido como Hello Word, donde partimos de una cadena de texto aleatoria y mediante aplicación de un algoritmo genético debemos obtener la cadena deseada.

1. Describa brevemente el diseño del cromosoma utilizado en el problema.

El programa define el alfabeto que va a utilizar y cómo son los genomas. En el programa también se define el cromosoma. Para acabar, define un conjunto inicial de determinados números de estados, que se conoce como la población inicial.

1. Describa brevemente la función de evaluación utilizada en el problema.

La función de evaluación que se utiliza es la variable fitnessValue, y se puede obtener su valor con el método getFitnessValue(). Este método cuenta el número de coincidencias que hay con el mensaje final que se desea obtener, y cuanto más valga esta variable más coincidencia hay con el cromosoma.

1. Ejecute el programa y realice un estudio sobre los resultados obtenidos con distinto número de poblaciones. Sólo es necesario modificar el valor de la variable **POBLACION** al inicio de la clase. (Para el análisis se aconseja el uso de Excel, la función *promedio*). A modo de ejemplo se incluye los datos con la población de un individuo.

**Población = 1**

|  |  |  |
| --- | --- | --- |
| **Ejecución** | **Generación Finaliza** | **Tiempo milisegundos** |
| 1 | 3164 | 381 |
| 2 | 6937 | 686 |
| 3 | 10083 | 1281 |
| 4 | 5823 | 688 |
| 5 | 8036 | 1005 |
| MEDIA | 6808 | 808 |

**Población = 10**

|  |  |  |
| --- | --- | --- |
| **Ejecución** | **Generación Finaliza** | **Tiempo milisegundos** |
| 1 | 410 | 461 |
| 2 | 703 | 997 |
| 3 | 573 | 818 |
| 4 | 545 | 673 |
| 5 | 861 | 976 |
| MEDIA | 618.4 | 785 |

**Población = 100**

|  |  |  |
| --- | --- | --- |
| **Ejecución** | **Generación Finaliza** | **Tiempo milisegundos** |
| 1 | 67 | 774 |
| 2 | 75 | 710 |
| 3 | 77 | 825 |
| 4 | 74 | 1016 |
| 5 | 49 | 533 |
| MEDIA | 68.4 | 771.6 |

**Población = 1000**

|  |  |  |
| --- | --- | --- |
| **Ejecución** | **Generación Finaliza** | **Tiempo milisegundos** |
| 1 | 31 | 1831 |
| 2 | 30 | 2037 |
| 3 | 28 | 1711 |
| 4 | 29 | 1873 |
| 5 | 31 | 1958 |
| MEDIA | 29.6 | 1882 |

**Conclusiones del experimento:**

Podemos observar que a mayor población, menos generaciones necesita para alcanzar la solución, en contrapartida el tiempo de cómputo que necesita es mucho mayor.

La mejor población de las que hemos probado en relación número de generaciones / tiempo es la de 100.

1. Ejecute el algoritmo ***manteniendo*** el operador CrossOver y **eliminando** el operador de mutación. A continuación analice el resultado tal y como se muestra a continuación. Utilice el método modifyConfiguration para establecer las distintas configuraciones. ***Modifique la información según el resultado de su ejecución***.

|  |  |  |
| --- | --- | --- |
|  |  |  |
|  |  |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Chr** | **Generación 1** | **Chr** | **Generación 2** |  |
| 1 | zUvs!WYuiuy KnshZQ (0) | 1 | **TiTSMyNruCS,h,ZtcP** (0) | **Cruce C3-C7** |
| 2 | zUvs!WYuiuy KnshZQ (0) | 2 | **c UcpOs,kjJvZVGiFA** (0) | **Cruce C7-C3** |
| 3 | **TiTSMyN,kjJvZVGiFA** (0) | 3 | zUvs!WYuiuy KnshZQ (0) | G1C1/C2/C10 |
| 4 | hujnHdsYzklCncZhER (0) | 4 | zUvs!WYuiuy KnshZQ (0) | G1C1/C2/C10 |
| 5 | ZbmHq!UniVTgqgMbym (0) | 5 | zUvs!WYuiuy KnshZQ (0) | G1C1/C2/C10 |
| 6 | YuGDYTAxkQ,hcTylUT (0) | 6 | TiTSMyN,kjJvZVGiFA (0) | G1C3 |
| 7 | **c UcpOsruCS,h,ZtcP** (0) | 7 | hujnHdsYzklCncZhER (0) | G1C4 |
| 8 | kPmPyVdAmgyNFCgVh (0) | 8 | ZbmHq!UniVTgqgMbym (0) | G1C5 |
| 9 | xwrElDxJjPipk!IzJG (0) | 9 | YuGDYTAxkQ,hcTylUT (0) | G1C6 |
| 10 | zUvs!WYuiuy KnshZQ (0) | 10 | **TiTSMyNruCS,h,ZtcP** (0) | **Cruce C3-C7** |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Chr** | **Generación 2** | **Chr** | **Generación 3** |  |
| 1 | TiTSMyNruCS,h,ZtcP (0) | 1 | **zUvs!WYuiuTgqgMbym** (0) | **Cruce C3-C8** |
| 2 | c UcpOs,kjJvZVGiFA (0) | 2 | **ZbmHq!UniVy KnshZQ** (0) | **Cruce C8-C3** |
| 3 | **zUvs!WYuiuy KnshZQ** (0) | 3 | TiTSMyNruCS,h,ZtcP (0) | G2C1/C10 |
| 4 | zUvs!WYuiuy KnshZQ (0) | 4 | c UcpOs,kjJvZVGiFA (0) | G2C2 |
| 5 | zUvs!WYuiuy KnshZQ (0) | 5 | TiTSMyNruCS,h,ZtcP (0) | G2C1/C10 |
| 6 | TiTSMyN,kjJvZVGiFA (0) | 6 | zUvs!WYuiuy KnshZQ (0) | G2C4/C5 |
| 7 | hujnHdsYzklCncZhER (0) | 7 | zUvs!WYuiuy KnshZQ (0) | G2C4/C5 |
| 8 | **ZbmHq!UniVTgqgMbym** (0) | 8 | zUvs!WYuiuy KnshZQ (0) | G2C4/C5 |
| 9 | YuGDYTAxkQ,hcTylUT (0) | 9 | TiTSMyN,kjJvZVGiFA (0) | G2C6 |
| 10 | TiTSMyNruCS,h,ZtcP (0) | 10 | **zUvs!WYuiuTgqgMbym** (0) | **Cruce C3-C8** |

**Conclusiones del experimento:**

¿Qué ocurre en la ejecución del programa sin la etapa de Mutación?

Que lo más probable es que la ejecución no termine ya que el algoritmo se puede quedar estancado en un máximo local.

Por ejemplo si en la población inicial no tenemos el carácter “!” del final de la frase que queremos obtener en este caso, sin la mutación nunca podremos obtenerlo.

Como inicialmente no tiene los caracteres que necesita, jamás podrá terminar. La mutación es necesaria.

**e)** Ejecute el algoritmo **eliminando** el operador CrossOver y **manteniendo** el operador de mutación. A continuación analice el resultado tal y como se muestra a continuación. Utilice el método modifyConfiguration para establecer las distintas configuraciones. ***Modifique la información según el resultado de su ejecución***.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Chr** | **Generación 1** | **Chr** | **Generación 2** |  |
| 1 | VGk!mFMBtinfNIXpTT (1) | 1 | V**W**k**RT**FMBtinfNIXpTT (1) | **G1C1-C10 Mutation** |
| 2 | UOultqzYVhLJTkACuM (0) | 2 | VGk!mFMBtinfNIXpTT (1) | G1C1/C10 |
| 3 | VGk!mFMByinfNIXgTT (0) | 3 | VGk!mFMBtinfNIXpTT (1) | G1C1/C10 |
| 4 | aTn!ZK ZPLDjKIIWwx (0) | 4 | aTn!ZK ZPLDjK**h**IMwx (0) | **G1C4 Mutation** |
| 5 | bo,tNcRlYFKkmwmaSf (0) | 5 | bo,tNcRlY**G**KkmwmaSf (0) | **G1C5 Mutation** |
| 6 | LNWnuFOGjfFQixAvES (0) | 6 | zDo,Mij**e**CPUxLxsCuM (0) | **G1C7 Mutation** |
| 7 | zDo,MijcCPUxLxsCuM (0) | 7 | UOultqzYVhLJTkA**J**uM (0) | **G1C2-C8 Mutation** |
| 8 | UOultqzYVhLJTkATuM (0) | 8 | lXZqBvpP!XoChZlXQa (0) | G1C9 |
| 9 | lXZqBvoP!XoChZlXQa (0) | 9 | UOultqzYVhLJTkACuM (0) | G1C2 |
| 10 | VGk!mFMBtinfNIXpTT (1) | 10 | V**W**k**RT**FMBtinfNIXpTT (1) | **G1C1-C10 Mutation** |

**Conclusiones del experimento:**

¿Qué ocurre en la ejecución del programa sin la etapa de CrossOver?

La ejecución acaba encontrando el patrón que buscaba aunque necesita de muchas generaciones puesto que la mutación es aleatoria y de un solo carácter.

Puede terminar ya que un carácter correcto no puede mutar, entonces solo es cuestión de tiempo que a cada carácter (de forma aleatoria) le toque el que corresponda y así pueda terminar