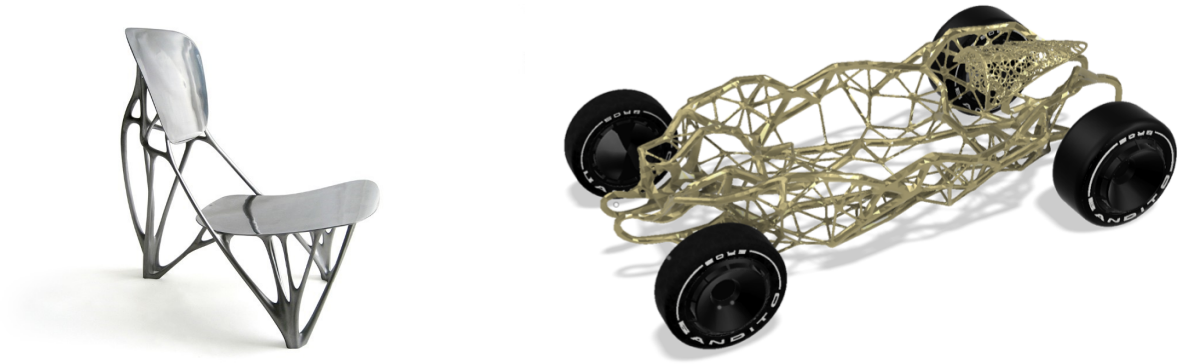
Tarea 2

Algoritmos Genéticos

1. Introducción

La evolución artificial se hace cada vez más presente en el mundo; a medida que aumenta la capacidad computacional disponible, ésta gana relevancia en todo tipo de áreas al permitir el desarrollo de inteligencia artificial, y la solución de distintos tipos de problemas. Existen distintas formas de aplicar evolución computacional a un sistema, pero todas tienen un principio en común, la inspiración en la evolución biológica; se replica la supervivencia del más fuerte mediante la modelación de individuos con código genético, los cuales pueden mutar y reproducirse, de acuerdo a una medida de su desempeño en el entorno.

En este contexto, los algoritmos genéticos representan solo una parte de la evolución computacional, pero se destacan por su relativa simplicidad, y utilidad en una gran variedad de campos. Los algoritmos genéticos son capaces de resolver una vasta cantidad de problemas, a veces obteniendo soluciones que sería virtualmente imposible ser concebidas por la mente humana (Figura 1.1); sin embargo, pueden ser superados en eficiencia por algoritmos especializados en problemas particulares, pero su alta capacidad de aplicación los convierte en una valiosa herramienta.



(a) Silla **(b)** Chasís de vehículo

Figura 1.1: Objetos con estructuras generadas por evolución computacional

Con el objetivo de introducir, y mostrar el poder de los algoritmos genéticos, se propone esta tarea, la cual consiste en 4 problemas para los que se crearan distintos algoritmos genéticos en Python, con el fin de solucionarlos y analizarlos.

2. Instalación

La tarea se desarrollará en Python, versión 2.7, la cual puede descargar en página oficial de Python: https://www.python.org/

Una vez instalado Python, se requiere descargar la librería *pyevolve*, la que posee funciones que serán de ayuda al crear un algoritmo genético. Para instalar *pyevolve*, se hace uso de la herramienta *Pip* de Python, que puede descargar librerías de Python por internet; pero primero, es recomendable actualizar *Pip*, lo que se puede hacer mediante el siguiente comando en CMD:

```
python -m pip install -U pip
```

Teniendo *Pip* actualizado, se instala *pyevolve* con:

```
python -m pip install pyevolve
```

El uso de un editor de texto facilita considerablemente la comprensión del código, ya que el IDE de Python no entrega una interfaz cómoda para trabajar. Un editor útil para esto es Sublime Text 3, el que puede descargar de la página oficial: https://www.sublimetext.com/

Finalmente, para acceder al directorio de un archivo mediante la ventana de comandos, lo que será necesario para ejecutar los archivos *.py*, puede ejecutar el comando *cd*, junto con la dirección de la carpeta donde se encuentra el archivo:

```
cd C:/Users/Lupita/Desktop/Tarea2
```

Y para ejecutar el archivo *p1.py*:

```
python p1.py
```

La tarea lleva adjunta archivos .py, para cada problema, que deberá usar para resolverlos.

En el anexo se encuentran todas las funciones de *pyevolve* que debería necesitar, junto con breves explicaciones. Si necesita más información sobre las funcionalidades de *pyevolve*, o ejemplos de uso, puede ver la documentación de esta librería en la página: http://pyevolve.sourceforge.net/

3. Problema 1

(2.2pts) Para comenzar, conviene familiarizarse con Python, y con el concepto de algoritmos genéticos; para esto, se propone un problema simple, que debe construir y solucionar, creando un algoritmo genético en Python, SIN la ayuda de librerías especializadas como *pyevolve*, guiándose del código del archivo *p1.py*

El problema se trata de encontrar el máximo de una clásica función cuadrática: $-x^2 + 100 \ \ / x \in [-10, 10]$ para esto, cada potencial solución, es decir, cada individuo de la población, debe representar un número REAL posible para x: y la forma más sencilla de hacer esto, es que cada individuo SEA un posible valor Real para x. Esto significa que:

- Individuo: Genoma de UN gen, Real entre $[-10, 10]$

Para el Crossover (Cruzamiento), una buena opción es crear un hijo tal que este sea el promedio aritmético entre los padres. Y para la mutación, que el valor del único gen del individuo cambie a un valor aleatorio entre $[-10, 10]$. Entonces:

- Crossover: Promedio aritmético
- Mutation: Valor random entre $[-10, 10]$

De los 3 tipos de selección comunes, uno conveniente para este problema es la selección por Torneo, aunque en general, el tipo de selección más usado es el de Ruleta.

- Selection: Torneo

Finalmente, la función de Fitness es simplemente calcular $-x^2 + 100$, siendo x el valor del individuo al que se le está calculando su fitness.

- Función Fitness: $-x^2 + 100$

En los archivos adjuntos se incluye el archivo *p1.py*, el que contiene una plantilla que le ayudará a crear el algoritmo, junto con instrucciones más específicas. Adjunte el archivo *p1.py* en su entrega.

1. ¿Qué resultado obtuvo el algoritmo?
2. ¿Es cercano al resultado esperado?

4. Problema 2

(1.2pts) Existen librerías especializadas en Python (y en otros lenguajes), que ayudan a la creación de algoritmos genéticos, entregando herramientas útiles que lo facilitan; este es el caso de *pyevolve*, que es una las tantas librerías disponibles para esto. El archivo adjunto *ex1.py*, muestra cómo se puede solucionar el problema 1, usando exactamente la misma representación, con *pyevolve*.

El problema anterior tiene distintas posibles representaciones, otra de ellas es representar el numero candidato a solución como un numero binario, es decir, como una lista de 0s y 1s que codifiquen un número real.

En esta ocasión, se intentará encontrar el máximo desplazamiento de una vibración mecánica con un grado de libertad (Figura 4.1), descrita por la función:

$$x(t) = 346e^{-0,0628t} \cdot sen(1,2544t + 2,35) \tag{4.1}$$

Usando una representación binaria de 4 bits para el tiempo, que limitaremos a $[0, 50]$. Note que tiene que aplicar una transformación lineal a los valores enteros que representan los bits, de modo que se ajuste al rango de $[0, 50]$ del problema.

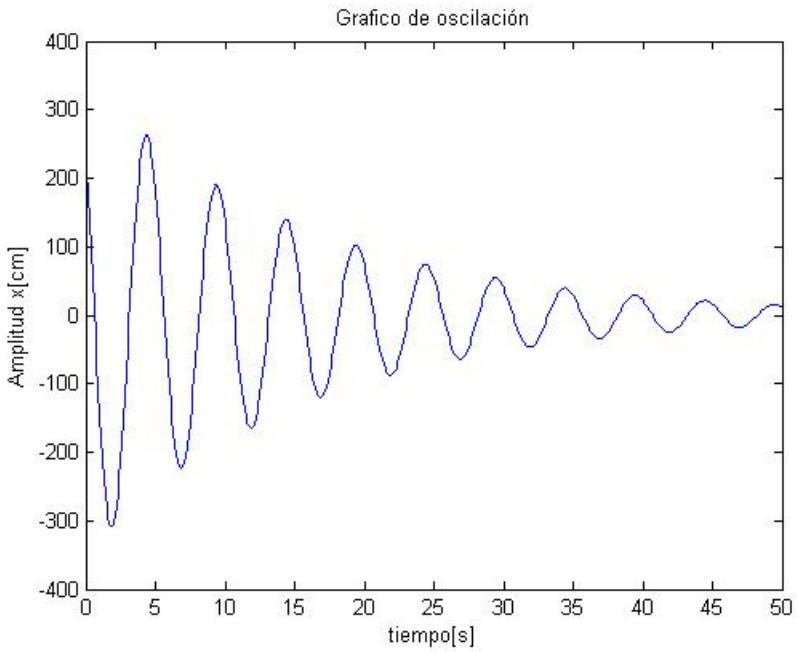


Figura 4.1: *Gráfico de la oscilación que representa la función 4.1*

Además, se puede notar que los métodos de Crossover y Mutation del problema anterior no tienen sentido en esta representación binaria, por lo que se hace necesario utilizar métodos distintos; en este caso, pueden ser de ayuda los métodos de Crossover uniforme, y Flip Mutator. Utilice selección por Ruleta (proporcional). Puede referirse a la sección de Anexo para buscar el nombre de las funciones de Inicialización, Mutación, Cruzamiento y Selección, de *pyevolve* que necesite para esto; puede ver el archivo *ex1.py* como ejemplo de uso.

En resumen, se necesita:

- Individuo: Genoma de 4 genes, Binario
- Crossover: Uniforme
- Mutation: Flip
- Selection: Ruleta (proporcional)
- Funcion Fitness: Función 4.1

En los archivos adjuntos se incluye el archivo *p2.py*, el que contiene una plantilla que le ayudará a crear el algoritmo usando *pyevolve*, junto con instrucciones más específicas. Puede usar el archivo *ex1.py* como guía extra. Adjunte el archivo *p2.py* en su entrega.

1. ¿Qué resultado obtuvo el algoritmo?
2. ¿Tiene sentido este resultado?
3. ¿Es un máximo local, o global?¿Es esto normal?¿Por qué?.Pruebe con un genoma de 10 bits.

5. Problema 3

(1.2pts)No solo números pueden formar parte de un genoma, en ocasiones, usar letras o cualquier otro tipo de dato u objeto, puede ser una mejor representación. Un genoma de ADN se compone de las proteínas A, C, T y G; y en computación el límite es muchísimo más extenso.

En este problema se usarán todos los caracteres del abecedario como alelos (alfabeto, posibilidades para gen), con el objetivo de que el algoritmo forme, por sí solo, la frase:

```
sometimes science is a lot more art than science
```

De esta manera, el genoma de un individuo corresponderá a una frase en sí, cada gen podrá tomar una letra del abecedario como valor, por lo que el conjunto de genes forma el intento de frase

Use un gen tipo lista, y las utilidades de la librería para usar alelos (o 'Alelle', en ingles) en el genoma, las que se encuentran detalladas en el archivo *p3.py*. Sumado a esto, en este caso conviene usar los métodos de mutación e inicialización definidos para trabajar con alelos, los que puede encontrar en el anexo.

Para hacer que los individuos se acerquen letra por letra al resultado óptimo, es decir que, si se quiere la letra 'h' en un gen, la letra 'g' valga más que la letra 'o', porque está más cerca a 'h' en el abecedario; se tiene que definir una distancia entre letras, que defina qué tan cerca o lejos se encuentra una letra de otra. Lo cual debe ser usado en la función de fitness, para dar un puntaje asociado a esta distancia. Un detalle importante es que el alfabeto sea considerado cíclico, que quiere decir esto, que la distancia entre 'a' y 'z', no sea 26 (por ser 26 letras), si no que 1, ya que después de 'z' puede volver al principio del alfabeto y comenzar por 'a'

Por último, pruebe el algoritmo usando 2 tipos de selección, Ranking y Ruleta, y compare los resultados. Utilice el archivo *p3.py*, y adjúntelo en su entrega.

1. ¿Qué resultado obtuvo el algoritmo con cada selector?
2. ¿Cual método de selección parece ser mejor para el problema? ¿Por qué podría ser así?

6. Problema 4

(1.4pts) Se puede usar un algoritmo genético para abordar el Problema del Vendedor Viajero, en este caso, considerando 20 ciudades a visitar. Suponga que se le entrega un mapa como el mostrado en la figura 6.1 y debe determinar una ruta que minimice el tiempo de viaje (o distancia recorrida), tal que pueda visitar todas las ciudades, sin visitar la misma ciudad más de una vez. La idea es no evaluar las 2432902008176640000 alternativas de viaje existentes y dejar que un Algoritmo Genético logre encontrar una buena solución con un número razonable de evaluaciones.

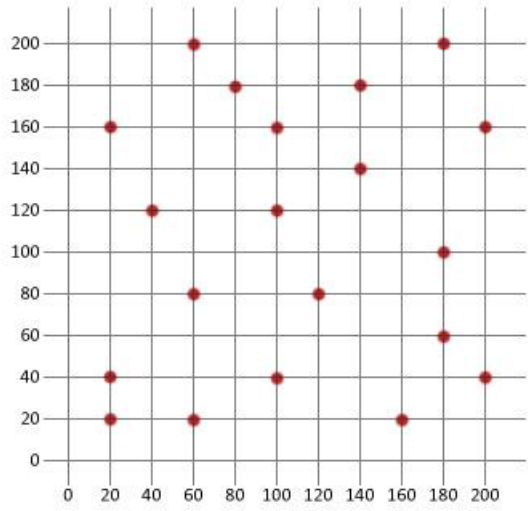


Figura 6.1: *Diagrama de ciudades a visitar*

Es necesario que las soluciones representadas sean válidas. Una ruta debe contener a todas las localizaciones en un orden sin repeticiones. Una ruta que contenga la misma localización repetida o que no tenga una localización no sería válida. Por lo que debe usar un genoma apropiado.

Respecto a los métodos de Mutation y Crossover, debido a que el problema se basa en el orden de visita de las ciudades, se requiere de métodos que modifiquen justamente eso, el orden. Para la mutación, un mecanismo que intercambie una ciudad por otra en el orden de visita. El cruzamiento se hace más complejo, se necesita uno que sea capaz de combinar el orden de los padres, y que no cree repeticiones de genes. Puede buscar ambos métodos en el Anexo.

Utilice el archivo *p4.py*, y adjúntelo en su entrega.

1. ¿Qué resultado obtuvo el algoritmo? Incluya una imagen con la ruta (aplique Paint)
2. ¿Parece optimo este resultado?
3. Incluya el gráfico de fitness máximo vs generacion, para distintos parámetros. Comente.

7. Anexo

7.1. Inicializadores

Initializators.G1DBinaryStringInitializator
Crea individuos tipo lista 1D con genes binarios aleatorios.

Initializators.G1DListInitializatorAllele
Crea individuos tipo lista 1D de alelos con genes aleatorios, donde el Alfabeto utilizado está definido por GAllele.GAlleleList().Para usar este inicializador, se deben especificar los alelos con las funciones GAllele.GAlleles.

Initializators.G1DListInitializatorInteger
Crea individuos tipo lista 1D con genes numéricos Enteros. Este inicializador acepta los parámetros rangemin y rangemax.

Initializators.G1DListInitializatorReal
Crea individuos tipo lista 1D con genes numéricos Reales. Este inicializador acepta los parámetros rangemin y rangemax.

7.2. Selectores

Selectors.GRankSelector
Selector de tipo Ranking.

Selectors.GRouletteWheel
Selector de tipo Ruleta (Proporcional).

Selectors.GTournamentSelector
Selector de tipo Torneo.

7.3. Mutadores

Mutators.G1DBinaryStringMutatorFlip
Cambia el valor de un gen de 0 a 1 o viceversa; toma genes aleatorios de un genoma G1DBinary.

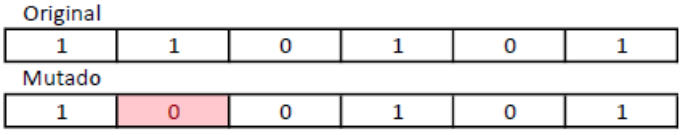


Figura 7.1: *G1DBinaryStringMutatorFlip*

Mutators.G1DBinaryStringMutatorSwap
Intercambia los valores de un gen por los del otro; toma genes aleatorios de un genoma G1DBinary.

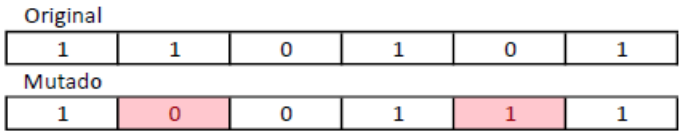


Figura 7.2: *G1DBinaryStringMutatorSwap*

Mutators.G1DListMutatorAllele
Cambia el valor de un gen por cualquiera de los valores definidos como alelos; toma genes aleatorios de un genoma G1DList. Para usar esta mutación, se deben especificar los alelos con las funciones GAllele.GAlleles. Toma genes aleatorios de un genoma G1DList.

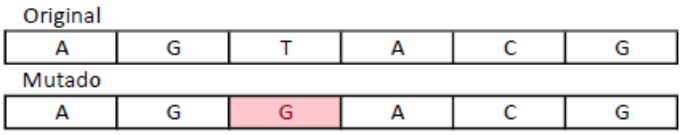


Figura 7.3: *G1DListMutatorAllele*

Mutators.G1DListMutatorIntegerRange
Cambia el valor de un gen por cualquier número Entero; toma genes aleatorios de un genoma G1DList. Esta mutación acepta los parámetros rangemin y rangemax. Toma genes aleatorios de un genoma G1DList.

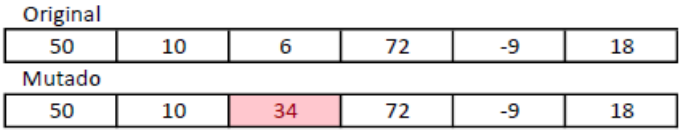


Figura 7.4: *G1DListMutatorIntegerRange*

Mutators.G1DListMutatorRealRange

Cambia el valor de un gen por cualquier número Real; toma genes aleatorios de un genoma G1DList. Esta mutación acepta los parámetros rangemin y rangemax. Toma genes aleatorios de un genoma G1DList.

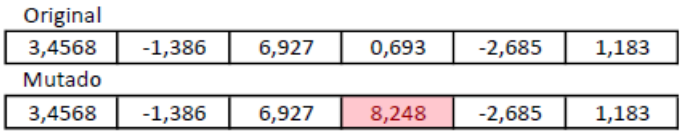


Figura 7.5: *G1DListMutatorRealRange*

Mutators.G1DListMutatorSwap

Intercambia los valores de un gen por los del otro; toma genes aleatorios de un genoma G1DList.

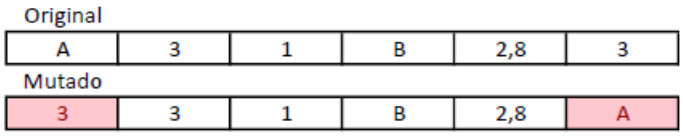


Figura 7.6: *G1DListMutatorSwap*

7.4. Cruzamientos

Crossovers.G1DBinaryStringXSinglePoint

Cruzamiento de un punto, para genes G1DBynary.

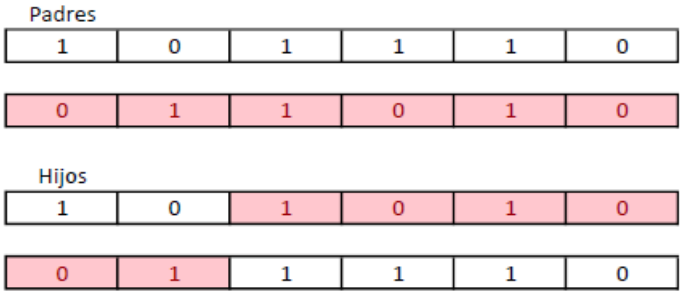


Figura 7.7: *G1DBinaryStringXSinglePoint*

Crossovers.G1DBinaryStringXTwoPoint
Cruzamiento de dos puntos, para genes G1DBynary.

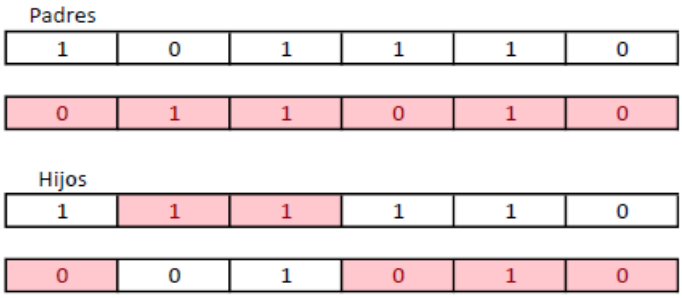


Figura 7.8: *G1DBinaryStringXTwoPoint*

Crossovers.G1DBinaryStringXUniform
Cruzamiento de todos los genes, para genes G1DBynary.

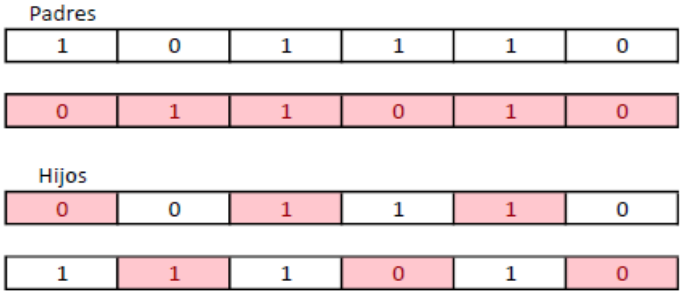


Figura 7.9: *G1DBinaryStringXUniform*

Crossovers.G1DListCrossoverOX
Cruzamiento donde se mantiene una porción del orden de uno de los padres, y se rellena el resto del genoma con el orden del otro, omitiendo los genes ya incluidos por el primer padre. Para genes G1DList.

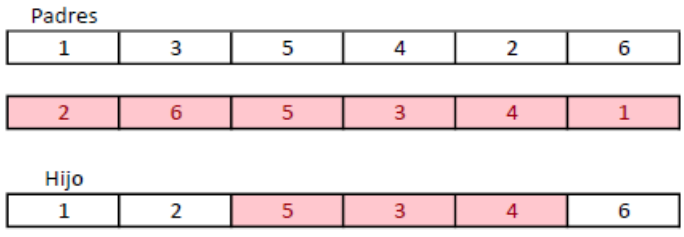


Figura 7.10: *G1DListCrossoverOX*

Crossovers.G1DListCrossoverSinglePoint

Cruzamiento de un punto, para genes G1DList.

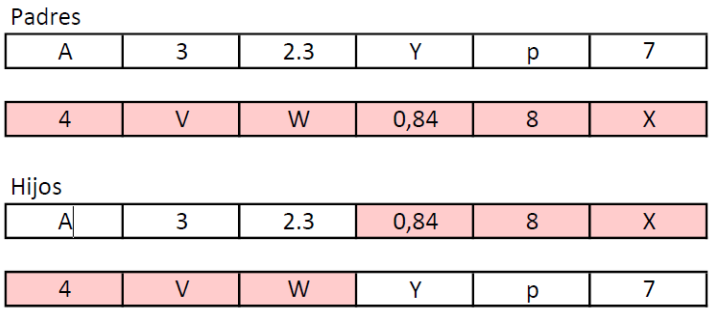


Figura 7.11: *G1DListCrossoverSinglePoint*

Crossovers.G1DListCrossoverTwoPoint
Cruzamiento de dos puntos, para genes G1DList.

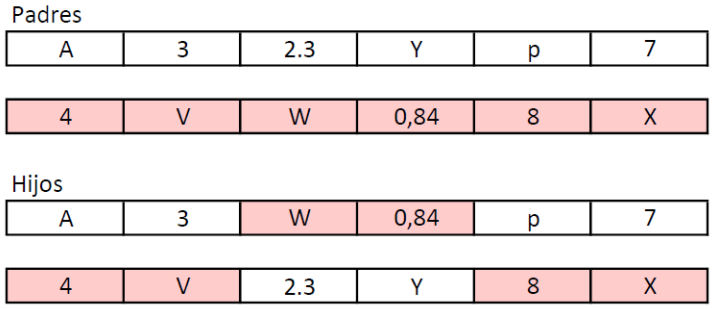


Figura 7.12: *G1DListCrossoverTwoPoint*

Crossovers.G1DListCrossoverUniform
Cruzamiento de todos los genes, para genes G1DList.

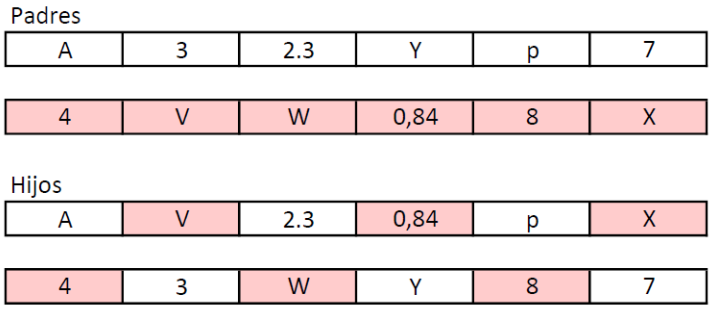


Figura 7.13: *G1DListCrossoverUniform*