Tarea 2 - Implementación de Algoritmo Genético

Profesor: Alexandre Bergel
Auxiliar: Juan-Pablo Silva
Ayudantes: Marco Caballero
María José Berger

Para esta tarea deberán implementar un algoritmo genético y aplicarlo a algún problema conocido. La idea es que utilicen el código desarrollado en clases a un problema donde no conozcan la solución, o no haya una solución obvia al problema.

1 Contenidos

En esta sección explicaremos lo que deberán hacer en la tarea y lo que se espera que tengan implementado.

1.1 Código del algoritmo

Como se describió en clases, un algoritmo genético viene descrito por 3 grandes etapas generales. Para esta tarea se espera que tenga estas 3 etapas implementadas, y un algoritmo funcional que logre explorar soluciones de manera evolutiva.

- **Evaluación:** debe existir una forma donde se pueda evaluar a cada individuo de la población, y así, *rankearlos*. Esta *evaluación* viene dada por la `función de fitness` descrita más abajo.
- **Selección:** debe existir una forma de seleccionar a los individuos de una población para elegir quiénes dejaran descendencia y quiénes no. Es importante considerar que debe existir diversidad dentro de nuestra población, ya que si no terminaremos con una población uniforme y podríamos caer en un mínimo/máximo local. Para la selección basta con que implemente alguno de los 2 algoritmos descritos en clases: la *ruleta* o el *torneo*.
- **Reproducción:** una vez seleccionados los individuos, deben existir operaciones de manipulación genética, llamadas *crossover* y *mutación*. Estas deben estar implementadas dentro del algoritmo, donde como mínimo debe proveer una operación de *crossover* (como la partición en algún punto al azar de un individuo), y una operación de mutación (como reemplazar un gen aleatorio dentro de un individuo). Para la mutación necesitara la `función de genes` descrita más abajo.

Para que el algoritmo funcione se deben proveer 3 funciones (en algunos casos 2 dependiendo de la implementación):

- **Función de *fitness*:** función que recibe 1 individuo y le asigna un valor, o conjunto de valores, numérico. Este valor se usa luego para la selección de quienes dejaran descendencia.

- **Función de generación de genes:** función que retorna un gen dependiente del problema a resolver. Podría ser un número, un `string`, una `tupla`, una `lista`, u otro objeto dentro de lo necesario de como se haya modelado el problema. Lo importante es que debe retornar un gen correspondiente a la decisión de diseño del planteamiento del problema.
- **Función de generación de individuos:** función opcional que crea un conjunto de genes, lo que se traduce a un individuo. El número de genes a generar viene determinado por ustedes.

Estas 3 funciones son dependientes del problema a resolver, por lo que no deben estar incluidas en el algoritmo, y deben sen pasadas como argumento al momento de iniciarlo.

Los valores de entrada para construir su algoritmo debe contener como mínimo las siguientes variables:

- **Tamaño de la población:** cuántos individuos se crearan inicialmente en el algoritmo. Este valor no implica que ustedes puedan aumentar o reducir la población después, eso depende de la implementación que quieran hacer.
- **Función de *fitness*:** explicado anteriormente, la función de *fitness* debe ser ingresada como argumento al algoritmo ya que es dependiente del problema.
- **Función de generación de genes:** igual que el *fitness*, es altamente dependiente del problema por lo que debe ser explicitada como argumento al algoritmo.
- **Tasa de mutación:** pueden interpretarlo como mejor crean, pero puede significar cuantos elementos de cada población variaran, o puede ser el porcentaje de genes de cualquier individuo que cambiara. Lo importante es que represente el porcentaje de mutación de un individuo o de un gen.
- **Condición de terminación:** puede ser un número máximo de iteraciones, o alguna condición más compleja, pero debe existir una condición de terminación para que el algoritmo no continúe para siempre. No puede usar el *"encontrar la solución"* como condición de terminación porque no siempre sabrá cuál es la solución al problema.

Por supuesto, puede tener más argumentos si lo estima necesario, como una función de generación de individuos, el número de genes, porcentaje de elitismo, entre otras.

1.2 Ejercicios

Para que su tarea sea válida, debe incluir los ejercicios elementales pedidos en clases. Estos son:

- **Secuencia de bits:** aquí debe considerar que tiene una lista, de tamaño fijo, que contiene solo `0`s y `1`s. También puede considerarlo con un `string` de `0`s y `1`s. La tarea es, dada una secuencia de bits, pedirle al algoritmo que encuentre la secuencia. Por ejemplo, puede ejecutar para encontrar la siguiente secuencia: `00101010110101`.

- **Encontrar una palabra/frase:** igual que el ejercicio de la secuencia de bits, pero ahora las opciones no son solo `0`s y `1`s, ahora pueden ser cualquier letra del abecedario. Puede intentar encontrar la frase: `helloworld`. Si quiere agregar dificultad, considere letras mayúsculas y espacios.

1.3 Elección de un problema

Aquí daremos 3 opciones para que elijan un problema donde aplicar el algoritmo genético:

- **Unbound-Knapsack:** Para una mochila que aguanta 15kg, encuentre la combinación de cajas que maximice el valor contenido en la mochila, pero que no se pase del peso máximo soportado. Las cajas permitidas son las siguientes:

  1. Caja de peso 12 y valor 4.
  2. Caja de peso 2 y valor 2.
  3. Caja de peso 1 y valor 2.
  4. Caja de peso 1 y valor 1.
  5. Caja de peso 4 y valor 10.

  Este problema se llama *Unbound* porque puede poner cuantas veces quiera cada caja, no hay límite (salvo por el límite de la mochila).

- **0-1-Knapsack:** Igual que el problema de *Unbound-Knapsack*, mismo peso máximo soportado por la mochila, y mismas cajas. La restricción es que solo tiene 1 caja de cada una, por lo que solo puede utilizar cada una a lo más 1 vez, no pueden haber repeticiones. No esta obligado a utilizarlas todas.

- **Laberinto:** Diseñe un laberinto aleatorio con obstáculos, una entrada y una salida. El objetivo es que el individuo parta en la entrada y deba llegar a la salida con el camino más corto.

Puede encontrar una referencia del problema *knapsack* o "de la mochila" en Wikipedia[1].

Para todos estos problemas usted esta en total libertad de como modelarlos, debe pensar una forma de representar cada problema tal que al algoritmo pueda hacer operaciones genéticas y de mutación sobre su representación.

1.4 Análisis

Debe generar 2 gráficos que permitan evaluar el comportamiento del algoritmo para algún problema en particular (usted puede elegir para qué problema, puede ser uno de los ejercicios). Los

[1] `https://en.wikipedia.org/wiki/Knapsack_problem`

gráficos que debe generar son los siguientes:

- **Mejora de *fitness* por generación:** debe ir guardando cómo cambian los *fitness* de los individuos a medida que avanzan las generaciones. En particular debe hacer un gráfico donde muestre como evoluciona el mejor de cada generación (el máximo de cada generación), el promedio de la generación y el peor de cada generación. De esta forma, podremos ver la historia de las generaciones utilizadas e identificar problemas en el proceso evolutivo, en caso de existir alguno.
- **Heatmap de configuraciones:** la idea del *Heatmap* es explorar exhaustivamente muchas combinaciones de **población** y de **tasa de mutación**, para encontrar la que mejor se adecua al problema. Esto se refiere a graficar, por ejemplo, para todas las combinaciones de la **población** en el rango $[50, \ldots, 1000]$ yendo de 50 en 50, y la **tasa de mutación** entre $[0.0, \ldots, 1.0]$ de 0.1 en 0.1. El cómo saber cuándo una combinación es "mejor" que otra se deja a su elección, pero puede considerar que si el algoritmo **demora mucho** en encontrar algo entonces es peor que otro que **demora menos**.

2 Evaluación

Para evaluar sus tareas se revisará su código, por lo que se pedirá que las partes relevantes estén comentadas para facilitar la corrección a los ayudantes. Además, para la parte de análisis, debe hacer un `readme` con esta información. A continuación se hace el desglose de evaluación:

- **Código del algoritmo** (2.0 puntos): se revisará que se haya implementado lo descrito en las secciones de contenido. Debe tener implementados los "ejercicios" descritos en la sección 1.2.
- **Problema elegido** (2.0 puntos): debe explicitar cuál es el problema se que busca resolver dentro de las opciones descritas en la sección 1.3. En el **readme** deberá indicar claramente qué considero como un gen, y qué como un individuo, además de la función de *fitness* que consideró. Cualquier información respecto a modificaciones al código para que se adapte a su problema también debe ir aquí.
- **Análisis** (2.0 puntos): haga una pequeña reflexión acerca de su implementación, que puede ser simplemente un comentario respecto a la utilidad del algoritmo o de las aplicaciones que le ve en la realidad basándose en el problema que resolvió. Debe, además, presentar y describir los gráficos que generó en la sección 1.4. Todo esto debe ser presentado en el `readme` de su repositorio.
  NO se extienda demasiado, es una descripción de lo que hizo más los resultados que obtuvo. Aquí puede encontrar una guía de como usar *markdown*[2].

[2] `https://github.com/adam-p/markdown-here/wiki/Markdown-Cheatsheet`

3 Entrega

La fecha de entrega de la tarea es el **lunes 14 de octubre**. Como usaremos Github para bajar sus tareas, basta con que en U-cursos suban **cualquier** archivo, y en los comentarios de entrega agreguen el enlace a su repositorio en Github.

Los análisis e imágenes de sus gráficos deben estar en el `readme` de su repositorio, no es necesario hacer un informe ni un documento en *pdf* explicando las cosas. La extensión en corta, solo debe hacer un pequeño reporte en el `readme` con los puntos señalados anteriormente.