Inteligencia artificial

Optimización con algoritmos genéticos

Reporte Algoritmo de Contenedores

Jesus Marcelo Torres Pineda 173207

Descripción del problema:

Se busca optimizar el envió de paquetes a través de contenedores. Estos paquetes tienen dos características, un peso y un valor. Los paquetes a enviar no deben de sobrepasar el tamaño del contenedor y se tiene entendido que mientras mas ganancias se den por contenedor, la solución es mas optima. Se debe mostrar un gráfico con la evolución de la ganancia de los paquetes enviados en los contenedores.

Configuración de software.

• Sistema operativo: Windows 10 Pro. 64 bits.

• Lenguaje: Python.

• Versión: 3.8.1

• Nombre del proyecto: Contenedores2A

• Enlace: github.com/JesusMarceloTorresPineda/Actividad2A

Modelado:

Iniciamos pidiendo 3 variables, “tamanioContenedo” para definir el tamaño del contenedor, “numPaquetes” para saber cuántos paquetes enviaremos, es necesario que sea número par ya que se eligen 2 individuos por generación y “paquetesIniciales” para saber con cuantos paquetes se iniciará la población.

Con dichas variables iniciamos la ejecución creando los paquetes que tendremos disponibles con el método “generate\_packages()”, con pesos del ‘1’ al ‘5‘ y generando un numero aleatorio de entre ‘1’ y ‘10’ le asignamos en valor a dicho paquete. Una vez asignados los valores a cada paquete, se agregan a una lista de nombre “packages” y es retornada por el método.

Despues procedemos a generar la población inicial con el método “generate\_population()” el cual recibe la variable “paquetesIniciales” ya antes definida, generamos un número random para saber que paquete llevara el individuo y se lo agregamos, para controlar que exceda el peso del contenedor vamos sumando ellos pesos de los paquetes que le agregamos al individuo y asi mientras que dicha suma sea menor al tamaño del contenedor, una vez no cumpla esa condición y sea valido el individuo lo agregamos a la lista de la población inicial, en caso contrario lo descartamos y se continua generando individuos hasta que todos salgan aceptados.

Ya generada la población inicial, esta se manda al método de “selección()”, en donde iteramos al arreglo de la población y calculamos su fitness y lo guardamos en el arreglo “aux\_fitness” y también guardamos a al individuo y a su fitness en otro arreglo llamado “evaluar\_esto” para tener referencia de sus fitness individuales. Al terminar de iterar la población se procede a sumar todos los fitness con el método “sum()” y calculamos el promedio de esa suma dividiéndolo en el número de individuos, después procedemos a calcular su probabilidad individual dividiendo los fitness individuales entre la suma de estos. Seguido, obtenemos el mejor y el peor individuo para posteriormente graficarlo, también se seleccionan los mejores individuos de la población y se retornan para continuar con el proceso.

Continuamos el proceso con el método “crossover()” que recibe a los individuos seleccionados y a partir de estos genera dos puntos de corte, “puntoCorte1” y “puntoCorte2”, uno para cada individuo. Obtenemos los valores de los individuos y los partimos en su respectivo punto de corte y los asignamos a dos variables auxiliares para unirlos despues. Generamos 3 arreglos, uno para cada hijo y otro para meter a los dos hijos. Procedemos a unir las cadenas separadas y las guardamos en su respectivo arreglo. Cuando cada hijo esta en su arreglo, estos se agregan al arreglo de hijos para validar que su peso no sea mayor al tamaño del contenedor con el método “validate\_weight()” que este itera cada hijo y suma sus costos, en caso de que alguno de ellos supere el tamaño del contenedor se vuelve a cruzar, partiendo de los mejores individuos de la generación y no de los ya cruzados hasta que encuentre un valor que si cumpla la condición, una vez cumpla la condición se agregan al arreglo “hijos\_returned” para devolverlos al método cruza. Esta comprobación la hace para cada hijo, en caso de que uno no supere el tamaño simplemente de agrega al arreglo.

Cuando ya se cruza y se hacen sus validaciones correspondientes, los mandamos a mutar al método “mutation()” donde iteramos la lista generada anteriormente y también se genera un aleatorio para indicarnos si debe mutar o no el individuo, en caso de que mutara, se itera cada paquete de la cadena y se hace una comprobación de manera aleatoria en la que si el paquete debe cambiar, es alterado en la misma posición, buscando así tener un valor diferente a la salida, con esto asegurando individuos que muten y otros que no. Aquí también despues de mutar validamos si esos individuos no exceden el tamaño del arreglo con el método “validate\_weight \_mutation()” que hace la suma correspondiente de los pesos de cada individuo, si este llega a sobrepasar la capacidad, se descarta y se vuelve a mutar hasta que se encuentre un individuo que si cumpla las condiciones o que se generen 30 iteraciones sin encontrar un candidato aceptable, ya que al ser aleatorio puede pasar que los valores excedan la capacidad y nunca encuentre un valor correcto, si no encuentra ninguno simplemente se devuelve el valor cruzado, en caso contrario, si el que muto es aceptado por la condición, este se agrega a su arreglo correspondiente y se devuelve al método “mutation()”.

Ahora con los nuevos individuos generados por todo el proceso, se integran al arreglo llamado “poblation” el cual contiene los dos mejores individuos previos y se agregan los dos nuevos, para comenzar nuevamente con el proceso de selección. Esto se repite hasta que el número de paquetes a enviar sea igual a un contador dentro del ciclo, ya que este contador controla el número de paquetes seleccionados por generación, es decir, los dos mejores de cada generación se agregan a los paquetes a enviar, y hasta obtener los resultados finales y generar una gráfica con las 3 curvas solicitadas, el mejor costo, el peor caso costo y el promedio por cada generación.

Selección:

El método de selección se encarga de seleccionar a los individuos con el costo más alto para enviarlos, primero se itera a la población y obtenemos su fitness individual y se guardan en un arreglo, posteriormente se busca en ese arreglo los mejores valores y estos son almacenados en otro arreglo para usarlos posteriormente.

Cruza:

El método de cruza genera una mezcla entre los individuos, buscando de esta manera generar hijos diferentes, para esto se itera el arreglo de los mejores individuos seleccionados, se generan 2 números aleatorios, uno para cada cadena, con esto sabremos su punto de corte, con eso proseguimos a cortar las cadenas y guardarlas en 2 variables por cada hijo, de esta manera aseguramos que no haya confusión al cruzarlas, despues se unen con su correspondiente y se agregan a dos listas distintas; “hijo1” e “hijo2”, para asi posteriormente meter esas 2 listas en otra llamada “hijitos”. Esta cadena resultante nos servirá para el método “validate\_weight()” el cual se asegura que cada hijo no sobrepase del peso del contenedor, en caso de ser asi, se genera otra cruza desde otro punto random para las cadenas originales hasta que se genere un individuo que satisfaga la condición.

Mutación:

El método de mutación se encarga de hacer iterar al arreglo de los individuos ya seleccionados y cruzados, y se genera un valor aleatorio que nos indica si ese individuo debe mutar o no. En caso de que el individuo mute, se itera a esa cadena de bits y nuevamente se hace la consulta para saber si cada bit debe ser mutado. De ser aceptada la modificación de dicho bit, este se cambiará de “0” a “1” o viceversa, de “1” a “0”, una vez terminado la comprobación de los bits e individuos, se integran al arreglo de individuos sustituyendo los valores anteriores.