Tarea Metaheurísticas”

“Metaheurísticas para 2-Opt. Consensus Problem for DNA Motif Profiles”

“Sistema Adaptativos”

Profesor: Pedro Pinacho Alumnos: Richard González, Vicente Cser y Vanessa Suazo

Pseudocódigo de la librería de búsqueda local que se hizo y metaheurística grasp.

Pseudocodigo de búsqueda local

```
Función createNeighbor(result: cadena) -> cadena
    neighbor = result
    index = aleatorio(0, longitud(result) - 1)
    list = ['A', 'C', 'T', 'G']
    randomIndex = aleatorio(0, 3)
    nucleotide = list[randomIndex]
    neighbor[index] = nucleotide
    retornar neighbor
Función acceptanceProbability(currentCost: entero, newCost:
entero, temperature: real) -> real
    Si newCost < currentCost Entonces
        retornar 1.0  // Acepta la nueva solución si es mejor.
    Sino
        retornar exp((currentCost - newCost) / temperature
Función calculateCost(result: cadena, dataset: vector de
cadenas) -> entero
    cost = 0
    distances = vector de enteros de longitud(dataset)
    // Rellenar el vector de distancias
    Para i desde 0 hasta longitud(result) - 1
        Para sequence desde 0 hasta longitud(dataset) - 1
            Si result[i] != dataset[sequence][i] Entonces
                distances[sequence]++
    // Calcular el costo
    Para cada value en distances
        cost += value * value
    retornar cost
Función localSearch(greedyResult: par de (entero, cadena),
dataset: vector de cadenas) -> par de (entero, cadena)
    currentCost = greedyResult.primerElemento
    currentResult = greedyResult.segundoElemento
    initialTemperature = 1000.0
    coolingRate = 0.99
    Mientras initialTemperature > 0.1 Hacer
        newResult = createNeighbor(currentResult)
        newCost = calculateCost(newResult, dataset)
        probability = acceptanceProbability(currentCost, newCost,
initialTemperature)
        Si probability > aleatorio(0, 1) Entonces
            currentResult = newResult
            currentCost = newCost
        initialTemperature *= coolingRate
    retornar par de (currentCost, currentResult)
```

Pseudocodigo de Grasp

```
Inicio del programa:
    Generar una semilla aleatoria para srand usando la hora actual
    Si el número de argumentos de línea de comandos (argc) es
mayor que 3:
        Si el primer argumento (argv[1]) no comienza con '-i' o no es
igual a "-i":
            Imprimir "Debes ingresar el comando -i"
            Salir del programa
    Sino:
        Imprimir "Debes ingresar el formato de ejecución <Greedy> -i
<instancia-problema> <probabilidad-aleatoriedad>"
        Salir del programa

    Abrir el archivo especificado en argv[2] para lectura
    Si el archivo se abre correctamente:
        Leer el contenido del archivo línea por línea y almacenarlo en
el vector dataset
        Cerrar el archivo
    De lo contrario:
        Imprimir un mensaje de error
        Salir del programa

    Inicializar una variable bestSolution
    Mientras (no se cumpla la condición de terminación):
        Ejecutar el algoritmo Greedy en el conjunto de datos
(dataset) con el valor de alpha
        Imprimir la solución inicial y su costo
        Ejecutar la búsqueda local en la solución generada por el
algoritmo Greedy y el conjunto de datos
        Imprimir la solución local y su costo
        Elegir la mejor solución
    Fin del bucle
    Finalizar el programa
```

La búsqueda local es un proceso de optimización que emplea cuatro funciones clave para modificar y evaluar soluciones de forma iterativa con el objetivo de encontrar una solución de menor costo. Estas funciones son las siguientes:

- **createNeighbor(result)**: Esta función crea una versión ligeramente modificada de una cadena de entrada llamada `result`. Realiza la modificación al cambiar aleatoriamente un carácter en la cadena.
- **acceptanceProbability(currentCost, newCost, temperature)**: Calcula la probabilidad de aceptar una nueva solución considerando las diferencias de costos entre la solución actual y la nueva, así como la temperatura actual. Si la nueva solución es mejor, se acepta con una probabilidad de 1.0.

- **calculateCost(result, dataset)**: Evalúa el costo de una solución al comparar la cadena `result` con un conjunto de cadenas en `dataset`. El costo se incrementa en función de cuántos caracteres difieren entre `result` y cada cadena del conjunto de datos.
- **localSearch(greedyResult, dataset)**: Esta función ejecuta una búsqueda local con el propósito de mejorar una solución inicial llamada `greedyResult`. Utiliza un algoritmo de enfriamiento simulado en el cual explora soluciones cercanas y decide si aceptarlas o rechazarlas basándose en la probabilidad de aceptación. La temperatura inicial disminuye gradualmente hasta alcanzar un valor muy bajo, lo que permite la exploración de soluciones cercanas en busca de una solución óptima o de menor costo.

La metaheurística Grasp implementada inicia inicializando una semilla aleatoria y verifica si se proporcionan argumentos de línea de comandos, donde se espera que el primer argumento sea "-i". Si los argumentos no cumplen con el formato requerido, el programa emite un mensaje de error y se detiene.
Cuando se proporcionan los argumentos correctamente, el programa intenta abrir un archivo especificado en el segundo argumento y lee su contenido línea por línea, almacenando en un vector llamado "dataset."
A continuación, el programa entra en un bucle, cuya condición de terminación no está definida en el pseudocódigo proporcionado. Dentro de este bucle, se ejecuta un algoritmo Greedy en los datos del "dataset" con un valor llamado "alpha," y se imprime la solución inicial y su costo.
Luego, se ejecuta una búsqueda local en la solución generada por el algoritmo Greedy, y se imprime la solución local y su costo.
Es importante notar que el bucle no tiene una condición de terminación definida, ya que está comentado con "//while(true)." Es necesario definir una condición de terminación adecuada para que el programa pueda finalizar de manera controlada.

La librería greedy.cpp es lo mismo greedy de la entrega anterior solo que modificado como una función pair que entrega costo y resultado que calcula greedy.
El manual básico se encuentra en el readme dentro del proyecto.