Tarea 1: Heurísticas Greedy para el 2-Optimality Consensus Problem

Sistemas Adaptativos

Segundo Semestre 2023, Prof. Pedro Pinacho

Alumno: Carlos Venegas Aguayo

1. Greedy Algorithm

```
function find_consensus():
    heuristic_cost = INT_MAX
    sequenceLength = length of dna motifs
    bestMotif = dna_motifs[0]

    for i = 0 to sequenceLength - 1:
        bestCharacter = dna_motifs[0][i]
        bestScore = heuristic_cost

        for candidateChar in "ATCG": //selecciona de caracteres
            currentScore = 0

            for candidate in dna_motifs:
                currentScore += calculateSquaredHammingDistance(bestMotif, candidate,
                i, candidateChar)

            if currentScore < bestScore:
                bestScore = currentScore
                bestCharacter = candidateChar
                heuristic_cost = bestScore

        bestMotif[i] = bestCharacter

    consensus_sequence = bestMotif
```

En este pseudo código se encuentra en resumen el algoritmo implementado, en el cual se utiliza el arreglo de strings considerados como DNA motifs, y se buscan para cada carácter la mejor letra candidata. Por cada letra candidata, se calcula el costo heurístico de reemplazarla para todas los DNA motifs y se elige la letra candidata con el menor costo heurístico. Este costo es dado por el cuadrado de las distancias de hamming, esto es básicamente:

```
function calculateSquaredHammingDistance(seq1, seq2, position, candidateChar):
    distance = 0
    for i = 0 to length of seq1 - 1:
        char1 = (i == position) ? candidateChar : seq1[i]
        char2 = seq2[i]

        if char1 != char2:
            distance = distance + 1

    return distance * distance
```

donde se va sumando 1 si los carácteres comparados son distintos, y finalmente se eleva al cuadrado la distancia obtenida.

A continuación se presentan los resultados de medias y desviación estándar de los tiempos de ejecución y costos heurísticos para los 100 archivos del dataset entregado por el profesor. Todo esto fue calculado mediante un script de PowerShell (GreedyHeuristic_average_std.ps1). Se provee en la entrega también un archivo similar para ejecutar en linux si es necesario(GreedyHeuristic_average_std.sh).

```
Average Heuristic Cost: 23518.4
Standard Deviation Heuristic Cost: 257.717325766042
Average Execution Time (milisegundos): 7.00788
Standard Deviation Execution Time (milisegundos): 4.09278106739171
```

2. Greedy Random Algorithm

Para la implementación de este algoritmo, se hizo algo muy similar al algoritmo Greedy original, con la diferencia de la introducción de un parámetro alpha. Este parámetro que va de 0 a 1, se utiliza para la selección de un carácter al azar [A,T,C,G] en vez de la selección normal greedy que se hace anteriormente, esto puede ser visto a continuación en el pseudo código:

```
function find_consensus():
    heuristic_cost = INT_MAX
    sequenceLength = length of dna motifs
    bestMotif = dna_motifs[0]

    for i = 0 to sequenceLength - 1:
        if random() < alpha:
            bestMotif[i]= getRandomCharacter()
        else:
            bestCharacter = dna_motifs[0][i]
            bestScore = heuristic_cost

            for candidateChar in "ATCG": //select from 4 characters
                currentScore = 0

                for candidate in dna_motifs:
                    currentScore += calculateSquaredHammingDistance(bestMotif,
                    candidate, i, candidateChar)
```

```
            if currentScore < bestScore:
                bestScore = currentScore
                bestCharacter = candidateChar
                heuristic_cost = bestScore

        bestMotif[i] = bestCharacter

    consensus_sequence = bestMotif
```

Donde se elige un carácter al azar si el numero generado random es menor que el parámetro alpha, de lo contrario el algoritmo actúa de la misma manera que el algoritmo greedy anterior.

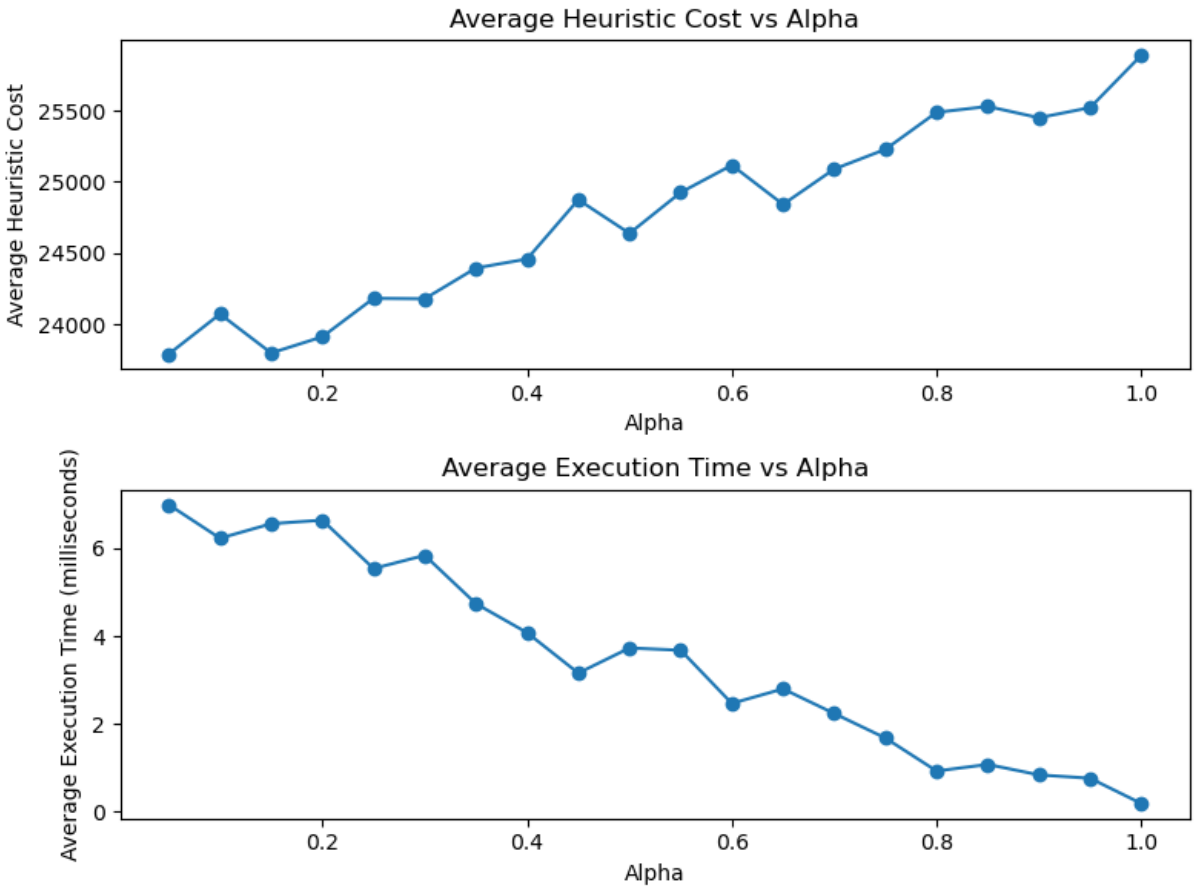

Figura 1: Como se puede ver en la primera figura, donde se tiene el costo promedio de la heurística. Se puede observar un crecimiento continuo mientras se aumenta el valor de alpha. Esto quiere decir que nos alejamos de la optimalidad de la solución encontrada al aumentar la aleatoriedad, por otro lado, se puede observar en el gráfico de abajo el tiempo de ejecución promedio vs el parámetro alpha; este gráfico nos indica que a medida que a medida que aumentamos la aleatoriedad del algoritmo disminuimos así también el tiempo de ejecución.

Es de concluir que junto con el algoritmo simple greedy y el algoritmo greedy random implementado, se tiene que se alcanza un mejor costo heurístico con el algoritmo determinístico pero a la vez con un mayor tiempo de ejecución.