INSTITUTO TECNOLÓGICO AUTÓNOMO DE MÉXICO

APLICACIÓN DE HEURÍSTICAS PARA LA RESOLUCIÓN DEL PROBLEMA DEL AGENTE VIAJERO

HECHO POR

MARIA ANTONIETA VALADEZ ROMERO 195697

MAXIMILIANO PÉREZ DE LA LLATA 200403

PROYECTO FINAL

[INVESTIGACIÓN DE OPERACIONES]

[Diciembre 2024]

ABSTRACT

El problema del agente viajero es uno de los problemas más estudiados en la optimización combinatoria debido a su relevancia en aplicaciones como la planificación logística y el diseño de redes de telecomunicaciones. Este trabajo presenta una comparación entre el algoritmo del vecino más cercano y un enfoque híbrido que combina dicho algoritmo con la heurística 2-opt, utilizando tres conjuntos de datos con coordenadas correspondientes a Qatar, Uruguay y Zimbabwe.

Los resultados muestran que, para instancias pequeñas o cuando el tiempo de ejecución es un factor crítico, el algoritmo del vecino más cercano es una solución eficiente. En cambio, para instancias más grandes o donde la calidad de la solución es prioritaria, el enfoque híbrido es más adecuado, ya que encuentra rutas de menor costo. Este trabajo resalta la importancia de elegir la estrategia adecuada dependiendo del balance requerido entre tiempo de ejecución y calidad de la solución.

INTRODUCCIÓN

El problema del agente viajero es uno de los más importantes dentro de la rama de la optimización combinatoria. Consiste en encontrar el camino más corto que permita visitar cada ciudad exactamente una vez y regresar a la ciudad de origen. La relevancia de este problema radica en sus aplicaciones en la planificación de proyectos logísticos y redes de telecomunicaciones, con el objetivo de optimizar los costos de transporte y reducir los tiempos de recorrido.

El problema del agente viajero tiene sus raíces en los trabajos de los matemáticos del siglo XIX, como el irlandés W. R. Hamilton y el británico Thomas Kirkman, quienes definieron conceptos fundamentales relacionados con las rutas y recorridos en grafos. Este problema, que consiste en encontrar la ruta más corta que un vendedor debe tomar para visitar un conjunto de ciudades y regresar al punto de inicio, ha evolucionado a lo largo del tiempo. En 1911, el matemático Karl Menger realizó importantes contribuciones a la teoría de grafos, sentando las bases para entender los recorridos más cortos en redes. A lo largo de la década de 1930, el problema comenzó a tomar forma más clara como un desafío de optimización, y matemáticos como P.K. Korte empezaron a aplicar la teoría de grafos a problemas de rutas eficientes. Estas ideas combinadas impulsaron el desarrollo de la investigación sobre el TSP, que hoy sigue siendo un tema central en la teoría de la optimización y la computación.

El problema del agente viajero puede modelarse como un grafo ponderado no dirigido, es decir, un grafo en el que las aristas tienen pesos y son bidireccionales. En este modelo, las ciudades se representan como vértices y las carreteras como aristas. Frecuentemente, el grafo es completo, lo que significa que todos los vértices están conectados mediante una arista. Sin embargo, si no existe un camino entre un par de vértices, se puede añadir una arista arbitraria para completar el grafo, sin afectar el recorrido óptimo.

MARCO TEÓRICO

El problema del agente viajero tiene una estrecha relación con el juego icosiano, desarrollado por Hamilton. En este juego, el objetivo es recorrer todos los vértices de un dodecaedro exactamente una vez y regresar al vértice de origen. Este juego permitió mostrar que no todos los grafos son hamiltonianos, es decir, no todos los grafos permiten recorrer todos sus vértices sin visitar ninguno más de una vez, ni sin tener que pasar por un vértice más de una vez. El problema del agente viajero, al igual que el juego icosiano, implica la búsqueda de una ruta óptima que recorra todos los puntos exactamente una vez, pero el problema del agente viajero se enfoca en encontrar la ruta más corta, lo que lo convierte en un problema de optimización.

El término ”problema del agente viajero” no apareció de inmediato, pero la formalización del problema en términos matemáticos se desarrolló en la década de 1950. Los matemáticos comenzaron a reconocer que el TSP (por sus siglas en inglés) era un problema NP-duro, es decir, que no se conoce una solución eficiente para grandes instancias y que la solución óptima requeriría un tiempo de cómputo que crece exponencialmente con el número de ciudades.

El algoritmo del vecino más próximo fue uno de los primeros métodos utilizados en ciencias de la computación para resolver el problema del agente viajero. Este algoritmo genera rápidamente una solución aproximada, aunque generalmente no es la óptima. Su funcionamiento consiste en elegir un vértice arbitrario como punto de inicio, luego encontrar la arista de menor peso conectada al vértice actual y a un vértice no visitado. Después, el vértice actual se convierte en el nuevo vértice visitado y se marca como tal. El proceso continúa hasta que todos los vértices han sido visitados, momento en el cual el algoritmo se cierra. Aunque es fácil de implementar y se ejecuta rápidamente, el algoritmo del vecino más

próximo puede perder rutas más cortas, que a menudo son evidentes para una evaluación visual humana.

El intercambio par a par o técnica 2-opt es una heurística que busca mejorar una solución inicial en el contexto del problema del agente viajero (TSP). En cada iteración, la técnica elimina dos aristas de la ruta actual y las reemplaza con dos nuevas aristas que reconectan los fragmentos resultantes tras la eliminación, produciendo así una nueva ruta que, en general, será más corta. El objetivo de este método es reducir el total de la distancia recorrida mediante la reconfiguración local de la ruta, eliminando cruces o redundancias.

Formulación del Problema del Agente Viajero (TSP) con Programación Lineal Entera

El *Problema del Agente Viajero (TSP)* puede ser formulado mediante *programación lineal entera*. Definimos x_{ij} como 1 si existe un camino de la ciudad i a la ciudad j, y 0 en caso contrario, para un conjunto de ciudades $0, \ldots, n$. Las variables u_i son artificiales, representando el orden en que se visitan las ciudades.

El modelo de programación lineal entera se plantea de la siguiente manera:

$$\min \sum_{i=0}^{n} \sum_{\substack{j \neq i \\ j=0}}^{n} c_{ij} x_{ij}$$

Sujeto a las siguientes restricciones:

$$0 \leq x_{ij} \leq 1, \quad x_{ij} \in \mathbb{Z} \quad \text{para } i, j = 0, \ldots, n$$

$$\sum_{\substack{i=0 \\ i \neq j}}^{n} x_{ij} = 1 \quad \text{para cada } j = 0, \ldots, n$$

$$\sum_{\substack{j=0 \\ j \neq i}}^{n} x_{ij} = 1 \quad \text{para cada } i = 0, \ldots, n$$

$$u_i - u_j + n x_{ij} \leq n - 1 \quad \text{para } 1 \leq i \neq j \leq n$$

Explicación del Modelo

- **Restricciones de salida y entrada**: Aseguran que cada ciudad tiene exactamente una salida y una entrada, garantizando que el recorrido cubra todas las ciudades.
- **Restricción de no sub-rutas**: La última restricción asegura que solo existe una secuencia cerrada de ciudades, evitando rutas disjuntas.

Prueba de la Validez de la Solución

1. **Secuencia cerrada**: Para que la solución sea válida, solo puede existir una secuencia cerrada de ciudades. Si se asumiera que hay sub-rutas que no pasan por la ciudad 0, se generaría una contradicción.
2. **Valores para las variables** u_i: Se define $u_i = t$ si la ciudad i es visitada en el paso t. Con esta asignación, las restricciones se satisfacen, asegurando que los valores de u_i estén dentro del rango válido y que se cumplan las restricciones para cada par de ciudades conectadas por una arista.

METODOLOGÍA

Vecino Más cercano

Este documento describe el funcionamiento de un código que implementa una solución heurística al problema del viajante utilizando el algoritmo del vecino más cercano.

1. Calcular distancia entre dos ciudades

Se calcula la distancia euclidiana entre dos puntos en el plano cartesiano.

- Entrada: dos ciudades representadas como tuplas (x, y).
- Proceso:
    1. Restar las coordenadas correspondientes de las dos ciudades.
    2. Elevar al cuadrado las diferencias obtenidas.
    3. Sumar los valores calculados.
    4. Calcular la raíz cuadrada del resultado.
- Salida: La distancia euclidiana entre las dos ciudades redondeada al entero más cercano.

Algorithm 1 Leer datos desde un archivo

```
function LeerDatos(archivo)
    ciudades ← []
    lineas ← LeerLineas(archivo)
    for linea ∈ lineas[1:] do                    ▷ Ignorar encabezado
        datos ← Separar(linea)
        if Longitud(datos) == 3 then
            x, y ← Convertir(datos[1], datos[2])
            Agregar(ciudades, (x, y))
        end if
    end for
    return ciudades
end function
```

2. Leer datos desde un archivo

El archivo de texto que contiene las coordenadas de las ciudades se lee una vez dado de alta en la carpeta.

- Entrada: Nombre del archivo con el formato siguiente:

```
Index  X      Y
0      0.0    0.0
1      1.0    1.0
2      2.0    2.0
```

- Proceso:

1. Abrir el archivo en modo lectura.
2. Ignorar la primera línea (encabezado).
3. Leer las líneas restantes y extraer las coordenadas (x, y) de cada ciudad.
4. Convertir las coordenadas a números flotantes y almacenarlas como una lista de tuplas (x, y).

- Salida: Una lista de tuplas (x, y) representando las coordenadas de las ciudades.

Algorithm 2 Leer datos desde un archivo

```
function LeerDatos(archivo)
    ciudades ← []
    lineas ← LeerLineas(archivo)
    for linea ∈ lineas[1 :] do                    ▷ Ignorar encabezado
        datos ← Separar(linea)
        if Longitud(datos) == 3 then
            x, y ← Convertir(datos[1], datos[2])
            Agregar(ciudades, (x, y))
        end if
    end for
    return ciudades
end function
```

3. Algoritmo del vecino más cercano La parte crucial del código, donde se encuentra una solución aproximada al TSP utilizando el vecino más cercano.

- Entrada: Lista de ciudades representadas como tuplas (x, y).
- Proceso:
    1. Iniciar en la primera ciudad y marcarla como visitada.
    2. Repetir hasta visitar todas las ciudades:
        (a) Encontrar la ciudad más cercana no visitada.
        (b) Marcar la ciudad como visitada.
        (c) Agregar la ciudad al recorrido.
    3. Regresar a la ciudad inicial para cerrar el recorrido.
- Salida: Una lista con el orden de las ciudades visitadas (recorrido) y la distancia total.

Algorithm 3 Algoritmo del vecino más cercano

```
1: function VecinoMasCercano(ciudades)
2:     n ← Longitud(ciudades)
3:     visitadas ← [Falso × n]
4:     recorrido ← [], distanciaTotal ← 0
5:     ciudadActual ← 0, visitadas[0] ← Verdadero
6:     Agregar(recorrido, 0)
7:     for i ← 1 hasta n − 1 do
8:         siguiente ← None, distanciaMinima ← ∞
9:         for j ← 0 hasta n − 1 do
10:            if ¬visitadas[j] then
11:                distancia ← CalcularDistancia(ciudades[ciudadActual], ciudades[j])
12:                if distancia < distanciaMinima then
13:                    distanciaMinima ← distancia
14:                    siguiente ← j
15:                end if
16:            end if
17:        end for
18:        visitadas[siguiente] ← Verdadero
19:        Agregar(recorrido, siguiente)
20:        distanciaTotal ← distanciaTotal + distanciaMinima
21:        ciudadActual ← siguiente
22:    end for
23:    distanciaTotal ← distanciaTotal+CalcularDistancia(ciudades[ciudadActual], ciudades[0
24:    Agregar(recorrido, 0)
25:    return (recorrido, distanciaTotal)
26: end function
```

4. Ejemplo de funcionamiento

- Archivo de entrada:

Index	X	Y
0	0.0	0.0
1	1.0	1.0
2	2.0	2.0
3	2.0	0.0

- Lista de ciudades leída: $[(0.0, 0.0), (1.0, 1.0), (2.0, 2.0), (2.0, 0.0)]$.
- Recorrido calculado:
  1. Inicio en $(0.0, 0.0)$.
  2. Ciudad más cercana: $(1.0, 1.0)$.
  3. Siguiente ciudad más cercana: $(2.0, 2.0)$.
  4. Última ciudad: $(2.0, 0.0)$.
  5. Regreso a $(0.0, 0.0)$.
- Resultado:
  - Recorrido: $[0, 1, 2, 3, 0]$.
  - Distancia total: Distancia calculada sumando todas las distancias del recorrido.

5. Notas finales

- Este algoritmo no garantiza la solución óptima del TSP.
- Es una heurística eficiente para problemas con muchas ciudades donde no se necesita precisión exacta.

Vecino más cercano y 2-opt: Heurística Híbrida

Este documento describe la implementación del algoritmo de **vecino más cercano** para resolver el problema del viajante (TSP), seguido de una mejora del recorrido obtenido utilizando la heurística **2-opt**.

1. Calcular distancia entre dos ciudades

Se calcula la distancia euclidiana entre dos puntos en el plano cartesiano.

- **Entrada:** Dos ciudades representadas como tuplas (x, y).
- **Proceso:**
  1. Restar las coordenadas correspondientes de las dos ciudades.
  2. Elevar al cuadrado las diferencias obtenidas.
  3. Sumar los valores calculados.
  4. Calcular la raíz cuadrada del resultado.
- **Salida:** La distancia euclidiana entre las dos ciudades redondeada al entero más cercano.

Algorithm 4 Calcular distancia euclidiana

```
function CALCULARDISTANCIA(ciudad1, ciudad2)
    dx ← ciudad1.x − ciudad2.x
    dy ← ciudad1.y − ciudad2.y
    distancia ← √(dx² + dy²)
    return round(distancia)
end function
```

2. Leer datos desde un archivo Se lee el archivo de texto que contiene las coordenadas de las ciudades

- **Entrada:** Archivo con las coordenadas de las ciudades.
- **Proceso:**
  1. Abrir el archivo y leer sus líneas, ignorando el encabezado.
  2. Para cada línea, extraer las coordenadas (x, y).
  3. Almacenar las coordenadas como una lista de tuplas (x, y).
- **Salida:** Lista de ciudades representadas como tuplas (x, y).

Algorithm 5 Leer datos desde un archivo

```
function LeerDatos(archivo)
    ciudades ← []
    lineas ← LeerLineas(archivo)
    for linea ∈ lineas[1 :] do                          ▷ Ignorar encabezado
        datos ← Separar(linea)
        if Longitud(datos) == 3 then
            x, y ← Convertir(datos[1], datos[2])
            Agregar(ciudades, (x, y))
        end if
    end for
    return ciudades
end function
```

3. Vecino Más Cercano Sección con el propósito de encontrar un recorrido aproximado utilizando la heurística del vecino más cercano.

- **Entrada:** Lista de ciudades representadas como tuplas (x, y).
- **Proceso:**
    1. Iniciar en una ciudad.
    2. Buscar iterativamente la ciudad no visitada más cercana.
    3. Agregarla al recorrido y marcarla como visitada.
    4. Continuar hasta visitar todas las ciudades.
    5. Regresar a la ciudad inicial para cerrar el recorrido.
- **Salida:** Recorrido aproximado y distancia total.

Algorithm 6 Vecino Más Cercano

```
1: function VECINOMASCERCANO(ciudades)
2:     n ← Longitud(ciudades)
3:     visitadas ← [Falso] × n
4:     recorrido ← [], distanciaTotal ← 0
5:     ciudadActual ← 0, visitadas[0] ← Verdadero
6:     Agregar(recorrido, 0)
7:     for i ← 1 hasta n − 1 do
8:         siguiente ← None, distanciaMinima ← ∞
9:         for j ← 0 hasta n − 1 do
10:            if ¬visitadas[j] then
11:                distancia ← CalcularDistancia(ciudades[ciudadActual], ciudades[j])
12:                if distancia < distanciaMinima then
13:                    distanciaMinima ← distancia
14:                    siguiente ← j
15:                end if
16:            end if
17:        end for
18:        visitadas[siguiente] ← Verdadero
19:        Agregar(recorrido, siguiente)
20:        distanciaTotal ← distanciaTotal + distanciaMinima
21:        ciudadActual ← siguiente
22:    end for
23:    distanciaTotal ← distanciaTotal+CalcularDistancia(ciudades[ciudadActual], ciudades[r
24:    Agregar(recorrido, 0)
25:    return (recorrido, distanciaTotal)
26: end function
```

4. Optimización 2-opt Es aquí donde se busca mejorar el recorrido dado mediante la heurística 2-opt.

- **Entrada:** Lista de ciudades y un recorrido inicial.
- **Proceso:**
    1. Iterar sobre todas las parejas de aristas en el recorrido.
    2. Intercambiar aristas si el cambio reduce la distancia total.
    3. Repetir hasta que no se puedan hacer más mejoras.
- **Salida:** Recorrido mejorado y distancia total optimizada.

Algorithm 7 Optimización 2-opt

```
function OPTIMIZAR2OPT(ciudades, recorrido)
    n ← Longitud(recorrido) − 1
    mejora ← Verdadero
    while mejora do
        mejora ← Falso
        for i ← 1 hasta n − 2 do
            for j ← i + 1 hasta n − 1 do
                if j − i == 1 then continuar
                end if
                Calcular la distancia antes y después del intercambio.
                if distancia mejorada then
                    Realizar el intercambio.
                    mejora ← Verdadero
                end if
            end for
        end for
    end while
    Calcular la distancia total.
    return (recorrido, distanciaTotal)
end function
```

ANÁLISIS DE RESULTADOS

El objetivo principal de esta comparación fue evaluar el desempeño y la eficiencia de dos enfoques heurísticos para resolver el Problema del Viajante (TSP): el algoritmo de Vecino Más Cercano y un enfoque híbrido que combina Vecino Más Cercano con la heurística 2-opt. Los experimentos se realizaron sobre tres conjuntos de datos con coordenadas de ciudades en Qatar, Uruguay y Zimbabwe. A continuación, se presenta el análisis de los resultados obtenidos.

En términos de tiempo de ejecución, el algoritmo de Vecino Más Cercano demostró ser significativamente más rápido que el enfoque híbrido. Para el caso de Qatar, el tiempo de ejecución fue prácticamente instantáneo (0.0 segundos), mientras que para Uruguay y Zimbabwe, el tiempo aumentó ligeramente a 0.3 y 0.4 segundos, respectivamente. Esto se debe a la simplicidad inherente de Vecino Más Cercano, ya que realiza únicamente comparaciones locales para determinar la ciudad más cercana en cada paso, sin realizar optimizaciones globales en el recorrido.

Por otro lado, el enfoque híbrido, que combina Vecino Más Cercano con 2-opt, presentó tiempos de ejecución considerablemente mayores, especialmente en los casos de Uruguay (6.3 segundos) y Zimbabwe (11.7 segundos). Esto ocurre porque la heurística 2-opt aplica mejoras iterativas al recorrido inicial generado por Vecino Más Cercano, intercambiando aristas para eliminar cruces y reducir el costo total del recorrido. Aunque este enfoque es computacionalmente más costoso, es capaz de generar soluciones de mayor calidad al optimizar globalmente el recorrido.

En cuanto a la calidad de las soluciones, el algoritmo de Vecino Más Cercano tiene la limitación de tomar decisiones locales, lo que puede llevar a rutas subóptimas si las elecciones iniciales no son ideales. Por el contrario, el enfoque híbrido utiliza el recorrido inicial como punto de partida y lo mejora mediante 2-opt, obteniendo rutas de menor costo en la mayoría de los casos. Este proceso de optimización es especialmente importante en instancias más grandes, donde las rutas generadas por Vecino Más Cercano pueden contener cruces innecesarios que aumentan significativamente el costo total.

Considerando la eficiencia y la escalabilidad, el algoritmo de Vecino Más Cercano es ideal para escenarios pequeños o cuando el tiempo de ejecución es un factor crítico,

ya que su simplicidad garantiza resultados rápidos, aunque no necesariamente óptimos. Por otro lado, el enfoque híbrido es más adecuado cuando la calidad de la solución es prioritaria, y el tiempo de cómputo no representa una restricción importante. Sin embargo, su escalabilidad es limitada debido al aumento significativo del tiempo de ejecución en instancias más grandes, como las de Uruguay y Zimbabwe.

CONCLUSIÓN

La combinación de las heurísticas del Vecino Más Cercano y el 2-opt ofrece un enfoque efectivo para resolver el problema del agente viajero. Mientras que el Vecino Más Cercano genera una solución inicial rápidamente, aunque subóptima, el 2-opt mejora significativamente la calidad del recorrido al eliminar intersecciones. Este enfoque combinado proporciona un equilibrio adecuado entre la calidad de la solución y el tiempo de ejecución, siendo una estrategia práctica para abordar problemas de optimización complejos. De esta forma, podemos mejorar la solución al problema del agente viajero, específicamente en el caso de encontrar la ruta óptima dentro de una ciudad. Si se requiere una solución más precisa, es posible combinar más de dos heurísticas para obtener un resultado más cercano al óptimo, aunque esto implique un mayor tiempo de resolución, ya que aumenta el costo computacional del cálculo del óptimo.

Bibliography

[1] Croes, G. A. (1958). ”A Method for Solving Traveling-Salesman Problems”. *Operations Research*, **6**(6), 791–800.

[2] Lawler, E. L., Lenstra, J. K., Rinnooy Kan, A. H. G., & Shmoys, D. B. (1985). *The Traveling Salesman Problem: A Guided Tour of Combinatorial Optimization*. Wiley.

[3] Cook, William (2011). *In Pursuit of the Travelling Salesman: Mathematics at the Limits of Computation*. Princeton University Press. ISBN: 978-0-691-15270-7.

[4] Gutin, G., & Punnen, A. P. (2002). *The Traveling Salesman Problem and Its Variations*. Springer.

APPENDIX: RESULTADOS

Apéndice 1: Resultado de la Ruta Menos Costosa con la Heurística del Vecino Más Cercano para Qatar

Recorrido:

0, 5, 7, 15, 12, 13, 95, 92, 96, 104, 105, 124, 125, 113, 112, 178, 173, 172, 174,
10, 16, 25, 23, 20, 106, 107, 109, 111, 108, 101, 102, 90, 183, 180, 176, 177,
17, 32, 27, 28, 21, 114, 115, 116, 120, 77, 74, 71, 73, 68, 179, 169, 166, 167,
26, 36, 38, 46, 50, 119, 127, 122, 123, 59, 56, 44, 83, 99, 164, 158, 157, 161,
57, 55, 52, 51, 47, 132, 134, 128, 130, 117, 121, 118, 110, 165, 170, 184, 192,
45, 40, 37, 39, 42, 135, 142, 147, 154, 103, 100, 98, 93, 89, 187, 190, 188, 191,
53, 54, 48, 41, 43, 150, 146, 151, 152, 88, 81, 79, 86, 75, 189, 186, 185, 182,
34, 31, 29, 30, 33, 149, 143, 153, 156, 70, 24, 22, 58, 61, 181, 175, 193, 159,
49, 60, 66, 65, 72, 140, 138, 137, 141, 35, 62, 64, 84, 85, 18, 14, 11, 9, 8, 4,
67, 63, 69, 76, 78, 145, 148, 144, 139, 97, 129, 155, 160, 2, 1, 3, 6, 19, 0
80, 82, 87, 91, 94, 136, 133, 131, 126, 162, 163, 168, 171,

Distancia total: 11640

Apéndice 2: Resultado de la Ruta Menos Costosa con el Enfoque Híbrido para Qatar

Recorrido optimizado:

0, 1, 3, 6, 2, 4, 8, 9, 76, 78, 80, 82, 87, 137, 138, 143, 140, 147, 159, 165, 170,
11, 14, 18, 29, 31, 92, 95, 94, 91, 96, 149, 153, 156, 152, 169, 166, 161, 157,
30, 33, 39, 42, 37, 72, 83, 99, 106, 107, 151, 146, 150, 154, 158, 164, 167, 177,
40, 43, 34, 49, 41, 104, 105, 117, 121, 135, 130, 128, 120, 179, 184, 192, 187,
48, 54, 53, 45, 47, 118, 125, 124, 126, 116, 115, 114, 111, 190, 188, 191, 189,
51, 52, 55, 57, 60, 131, 133, 136, 139, 109, 119, 123, 122, 193, 175, 181, 186,
66, 65, 67, 63, 69, 144, 148, 145, 141, 127, 132, 134, 142, 185, 182, 183, 180,

176, 174, 172, 173, 35, 61, 58, 22, 24, 108, 101, 102, 90, 27, 32, 17, 20, 23,
178, 171, 168, 163, 70, 75, 86, 79, 81, 77, 74, 71, 73, 68, 25, 16, 10, 13, 12,
162, 160, 155, 129, 88, 89, 93, 98, 100, 59, 56, 44, 36, 50, 15, 7, 5, 19, 0
97, 85, 84, 64, 62, 103, 110, 113, 112, 46, 38, 26, 21, 28,

Distancia optimizada: 10539

Apéndice 3: Resultado de la Ruta Menos Costosa con la Heurística del Vecino Más Cercano para Uruguay

Recorrido:

0, 2, 5, 10, 21, 17, 363, 367, 391, 389, 459, 471, 475, 489, 235, 233, 204, 125,
15, 26, 46, 45, 44, 410, 423, 446, 445, 499, 493, 484, 492, 119, 120, 123, 122,
35, 24, 20, 18, 16, 9, 447, 450, 438, 437, 507, 498, 497, 478, 99, 94, 82, 65, 55,
7, 3, 4, 6, 1, 14, 11, 421, 415, 409, 417, 469, 464, 453, 461, 52, 49, 43, 53, 62,
12, 13, 8, 22, 23, 25, 407, 441, 435, 457, 479, 503, 511, 516, 66, 67, 75, 74, 73,
27, 33, 39, 38, 42, 474, 485, 506, 529, 525, 535, 522, 571, 72, 71, 57, 104, 108,
50, 68, 59, 54, 37, 537, 532, 533, 540, 590, 613, 622, 632, 109, 117, 132, 126,
30, 28, 19, 41, 48, 551, 572, 591, 598, 543, 442, 425, 418, 114, 90, 85, 80, 79,
64, 61, 69, 88, 89, 605, 614, 618, 623, 386, 371, 383, 373, 78, 84, 83, 92, 76,
111, 106, 118, 130, 615, 609, 603, 587, 374, 347, 337, 336, 40, 91, 97, 112, 103,
136, 141, 139, 133, 585, 606, 576, 580, 321, 324, 325, 312, 107, 70, 56, 51, 58,
131, 124, 142, 146, 561, 570, 577, 599, 316, 295, 294, 268, 60, 47, 36, 34, 32,
150, 162, 160, 156, 600, 610, 583, 578, 261, 269, 262, 280, 29, 31, 63, 81, 96,
159, 155, 166, 167, 574, 588, 579, 611, 288, 255, 245, 246, 116, 113, 110, 87,
178, 177, 175, 158, 616, 620, 589, 566, 256, 241, 239, 228, 129, 138, 154, 174,
149, 185, 198, 206, 601, 629, 612, 621, 208, 212, 181, 180, 165, 188, 202, 225,
199, 232, 234, 250, 617, 631, 584, 575, 191, 183, 195, 186, 240, 263, 275, 259,
242, 238, 243, 230, 557, 558, 550, 542, 194, 190, 179, 164, 272, 303, 343, 358,
226, 219, 223, 273, 538, 526, 517, 508, 161, 168, 169, 170, 362, 372, 405, 404,
283, 276, 266, 267, 512, 518, 521, 519, 211, 214, 236, 252, 416, 429, 432, 430,
271, 279, 304, 300, 514, 500, 480, 472, 244, 265, 293, 311, 436, 456, 476, 486,
329, 335, 339, 331, 482, 462, 448, 454, 305, 264, 260, 251, 501, 504, 490, 483,

505, 520, 524, 549, 560, 586, 597, 625, 635, 634, 638, 647, 655, 669, 679, 683, 688, 672, 676, 668, 656, 659, 673, 681, 689, 698, 697, 699, 702, 706, 713, 716, 726, 731, 732, 728, 727, 729, 730, 718, 717, 712, 709, 700, 694, 695, 704, 703, 707, 708, 720, 719, 714, 696, 690, 686, 657, 654, 645, 644, 643, 650, 665, 671, 677, 684, 687, 701, 705, 710, 721, 715, 724, 725, 685, 680, 653, 637, 624, 626, 627, 628, 604, 608, 630, 639, 642, 662, 658, 649, 663, 664, 670, 641, 636, 563, 547, 510, 502, 487, 491, 488, 513, 556, 559, 555, 541, 534, 536, 452, 458, 394, 380, 364, 351, 330, 342, 291, 207, 189, 247, 270, 313, 356, 401, 411, 420, 378, 352, 400, 477, 496, 554, 569, 573, 562, 619, 646, 648, 661, 675, 674, 682, 693, 678, 667, 660, 640, 602, 595, 568, 581, 596, 582, 652, 692, 723, 722, 733, 711, 691, 651, 633, 565, 523, 531, 546, 553, 527, 495, 473, 468, 470, 444, 427, 428, 419, 408, 388, 375, 376, 379, 382, 346, 332, 310, 298, 289, 286, 285, 282, 274, 257, 253, 249, 227, 221, 213, 187, 184, 192, 196, 205, 215, 171, 163, 157, 143, 135, 115, 127, 128, 140, 144, 147, 151, 176, 172, 201, 210, 216, 217, 222, 224, 218, 231, 237, 254, 258, 287, 307, 319, 306, 281, 302, 314, 326, 318, 308, 317, 296, 292, 284, 277, 278, 248, 220, 209, 203, 301, 357, 360, 387, 392, 395, 396, 406, 397, 384, 402, 414, 439, 451, 463, 460, 467, 466, 494, 548, 544, 545, 552, 564, 593, 607, 592, 594, 539, 530, 515, 481, 465, 455, 443, 424, 403, 393, 390, 398, 381, 366, 365, 361, 348, 353, 349, 355, 426, 440, 433, 431, 422, 413, 412, 385, 368, 359, 338, 327, 328, 320, 323, 315, 297, 290, 309, 322, 333, 341, 350, 340, 354, 299, 344, 345, 334, 370, 377, 369, 399, 434, 449, 509, 528, 567, 666, 200, 197, 182, 193, 173, 152, 153, 145, 137, 121, 105, 93, 100, 101, 86, 77, 95, 98, 102, 134, 148, 229, 0

Distancia total: 992

Apéndice 4: Resultado de la Ruta Menos Costosa con el Enfoque Híbrido para Uruguay

Recorrido optimizado:

0, 2, 5, 10, 21, 35, 44, 57, 67, 66, 75, 74, 73, 62, 53, 52, 49, 43, 40, 24, 20, 18, 16, 1, 6, 4, 3, 7, 9, 14, 11, 8, 13, 12, 22, 23, 25, 27, 33, 39, 38, 42, 50, 68, 59, 54, 37, 30, 28, 19, 41, 48, 56, 51, 58, 60, 47, 36, 34, 32, 29, 31, 63,

81, 96, 87, 110, 113, 506, 529, 537, 532, 360, 384, 397, 406, 687, 701, 706, 702,
116, 129, 138, 154, 533, 540, 496, 477, 396, 395, 392, 387, 699, 697, 689, 681,
148, 134, 102, 98, 487, 502, 510, 547, 402, 414, 439, 451, 673, 659, 653, 637,
95, 77, 86, 93, 100, 563, 585, 587, 603, 463, 460, 467, 466, 624, 626, 627, 628,
101, 105, 121, 137, 609, 615, 623, 618, 494, 548, 544, 545, 604, 608, 630, 639,
145, 153, 152, 173, 614, 605, 598, 591, 552, 564, 592, 607, 642, 662, 658, 670,
193, 182, 201, 210, 572, 551, 554, 569, 593, 594, 539, 530, 664, 663, 649, 641,
216, 217, 218, 224, 573, 562, 619, 646, 515, 481, 465, 426, 636, 606, 576, 580,
222, 231, 237, 200, 648, 661, 679, 669, 365, 361, 348, 353, 561, 570, 577, 599,
197, 172, 176, 151, 655, 647, 625, 597, 349, 366, 381, 398, 600, 610, 583, 578,
147, 144, 140, 128, 586, 528, 509, 449, 390, 393, 403, 443, 574, 588, 579, 611,
127, 115, 135, 143, 434, 399, 369, 377, 455, 424, 419, 428, 616, 620, 589, 566,
157, 163, 171, 184, 370, 344, 345, 334, 427, 444, 470, 468, 571, 601, 629, 631,
192, 196, 205, 215, 299, 297, 290, 309, 473, 495, 527, 553, 617, 621, 612, 584,
187, 213, 221, 227, 322, 333, 340, 354, 546, 531, 523, 565, 575, 632, 622, 613,
249, 253, 257, 274, 341, 350, 315, 323, 633, 651, 666, 691, 590, 557, 558, 550,
282, 285, 289, 286, 320, 328, 327, 338, 711, 733, 722, 723, 542, 543, 538, 526,
254, 258, 287, 298, 359, 368, 385, 412, 692, 678, 667, 660, 517, 508, 512, 518,
310, 332, 346, 362, 413, 422, 431, 433, 640, 638, 634, 635, 521, 519, 514, 500,
358, 343, 303, 272, 430, 432, 429, 436, 652, 674, 682, 693, 480, 472, 482, 462,
259, 275, 263, 240, 456, 476, 486, 501, 675, 683, 688, 672, 448, 420, 378, 401,
225, 202, 174, 165, 504, 490, 483, 505, 676, 668, 656, 680, 411, 454, 459, 471,
188, 206, 229, 232, 520, 524, 549, 560, 685, 698, 715, 721, 511, 516, 525, 535,
234, 250, 242, 238, 582, 596, 581, 602, 710, 705, 724, 725, 522, 503, 499, 493,
243, 273, 283, 276, 595, 568, 567, 440, 713, 716, 726, 731, 484, 492, 507, 536,
266, 267, 271, 279, 416, 404, 405, 372, 732, 728, 727, 729, 534, 556, 559, 555,
304, 300, 329, 335, 382, 379, 376, 375, 730, 718, 717, 712, 541, 513, 488, 491,
339, 331, 363, 367, 388, 408, 355, 307, 709, 700, 694, 695, 458, 452, 498, 497,
391, 389, 410, 423, 319, 306, 281, 302, 704, 703, 707, 708, 478, 469, 464, 453,
446, 450, 447, 445, 314, 326, 318, 308, 720, 719, 714, 696, 461, 479, 489, 475,
438, 437, 421, 415, 278, 248, 203, 209, 690, 686, 657, 654, 442, 425, 418, 383,
409, 417, 407, 441, 220, 277, 284, 292, 645, 644, 643, 650, 371, 386, 400, 394,
435, 457, 474, 485, 296, 317, 301, 357, 665, 671, 677, 684, 380, 364, 351, 330,

342, 311, 305, 264, 260, 251, 235, 233, 204, 219, 226, 230, 223, 199, 198, 185, 149, 158, 175, 177, 178, 167, 166, 155, 159, 156, 160, 162, 150, 146, 142, 124, 131, 133, 139, 141, 136, 130, 118, 106, 111, 89, 88, 69, 61, 64, 70, 107, 103, 112, 97, 91, 94, 82, 65, 55, 71, 72, 99, 122, 123, 120, 119, 125, 207, 244, 265, 293, 291, 252, 236, 268, 294, 295, 316, 312, 336, 337, 352, 373, 374, 347, 324, 321, 325, 356, 313, 270, 247, 241, 256, 255, 288, 280, 261, 269, 262, 245, 246, 239, 228, 208, 212, 181, 180, 191, 214, 211, 186, 195, 183, 170, 169, 168, 190, 194, 189, 179, 164, 161, 104, 108, 109, 117, 132, 126, 114, 90, 85, 80, 79, 78, 84, 83, 92, 76, 45, 46, 26, 17, 15, 0

Distancia optimizada: 88796

Apéndice 5: Resultado de la Ruta Menos Costosa con la Heurística del Vecino Más Cercano para Zimbabwe

Recorrido:

0, 8, 6, 11, 16, 25, 22, 23, 24, 31, 43, 62, 39, 40, 41, 21, 45, 48, 81, 85, 84, 83, 87, 93, 92, 236, 254, 296, 327, 343, 346, 345, 348, 355, 357, 377, 381, 385, 392, 393, 394, 396, 405, 398, 408, 409, 410, 416, 422, 423, 431, 430, 429, 441, 454, 462, 448, 463, 471, 455, 480, 492, 505, 498, 499, 506, 518, 527, 532, 541, 540, 547, 556, 568, 569, 557, 577, 585, 579, 580, 586, 590, 603, 608, 607, 598, 589, 594, 593, 584, 576, 571, 555, 546, 536, 531, 504, 491, 470, 460, 459, 479, 490, 489, 497, 488, 478, 458, 453, 447, 469, 468, 457, 446, 452, 456, 475, 485, 476, 486, 502, 512, 516, 511, 510, 509, 495, 484, 473, 466, 451, 439, 426, 433, 443, 438, 434, 444, 445, 467, 474, 496, 515, 521, 522, 534, 537, 543, 544, 538, 535, 524, 523, 530, 539, 545, 551, 565, 564, 563, 559, 550, 508, 552, 553, 560, 554, 561, 566, 567, 583, 600, 604, 605, 601, 614, 622, 618, 617, 599, 581, 582, 623, 624, 661, 743, 685, 662, 726, 777, 794, 635, 633, 615, 620, 619, 611, 606, 602, 572, 595, 609, 634, 663, 686, 843, 853, 863, 854, 857, 861, 870, 881, 890, 893, 898, 903, 904, 906, 909, 910, 912, 915, 917, 911, 916, 907, 913, 918, 920, 922, 927, 925, 924, 923, 921, 926, 928, 919, 914, 908, 905, 901, 900, 896, 886, 879, 880, 874, 885, 889, 891, 892, 888, 869, 866, 856, 849, 855, 833, 825, 796, 759, 638, 625, 631, 649, 621, 575, 425, 483, 548, 542, 529, 494, 465, 412, 424,

437, 514, 472, 482, 53, 55, 50, 44, 38, 184, 181, 172, 152, 782, 785, 784, 775,
481, 501, 442, 436, 37, 27, 32, 36, 46, 189, 201, 210, 223, 786, 797, 808, 809,
411, 390, 380, 371, 59, 72, 88, 96, 102, 209, 195, 183, 197, 806, 805, 815, 807,
368, 350, 351, 353, 132, 251, 261, 256, 212, 218, 220, 231, 792, 790, 788, 740,
361, 356, 320, 316, 273, 272, 293, 306, 219, 191, 170, 150, 752, 748, 737, 734,
310, 294, 299, 298, 297, 304, 303, 309, 112, 111, 121, 107, 724, 718, 711, 699,
302, 288, 278, 289, 315, 325, 329, 314, 106, 95, 98, 229, 693, 694, 684, 681,
286, 266, 243, 248, 318, 270, 285, 269, 222, 214, 199, 227, 676, 673, 669, 659,
245, 249, 247, 246, 260, 264, 265, 271, 233, 180, 203, 237, 656, 657, 674, 666,
242, 240, 239, 238, 281, 131, 78, 67, 65, 153, 204, 101, 97, 670, 668, 677, 642,
241, 232, 224, 230, 51, 26, 18, 12, 14, 90, 94, 75, 61, 58, 658, 683, 702, 697,
226, 235, 228, 216, 20, 33, 35, 28, 19, 70, 276, 301, 300, 713, 715, 728, 701,
213, 198, 188, 177, 29, 17, 13, 30, 42, 290, 295, 311, 319, 689, 688, 696, 698,
175, 171, 168, 162, 47, 56, 57, 49, 76, 317, 328, 312, 349, 703, 710, 720, 712,
157, 155, 140, 144, 71, 64, 79, 178, 287, 374, 364, 369, 334, 727, 731, 739, 735,
151, 148, 141, 137, 308, 307, 330, 332, 323, 578, 612, 637, 733, 746, 751, 738,
139, 136, 134, 135, 338, 347, 359, 362, 647, 648, 754, 816, 730, 722, 714, 705,
142, 149, 158, 169, 365, 354, 358, 383, 744, 839, 848, 852, 717, 709, 695, 691,
167, 163, 160, 166, 507, 519, 520, 533, 820, 646, 636, 630, 682, 675, 690, 692,
164, 138, 128, 116, 597, 745, 822, 821, 687, 616, 632, 613, 700, 708, 719, 707,
129, 113, 123, 114, 819, 829, 824, 840, 570, 574, 592, 591, 704, 721, 732, 741,
119, 103, 125, 127, 828, 831, 836, 837, 629, 795, 729, 838, 736, 761, 755, 772,
122, 115, 110, 117, 842, 862, 876, 882, 844, 858, 868, 875, 781, 778, 773, 780,
118, 126, 124, 120, 895, 902, 867, 860, 894, 899, 883, 872, 769, 764, 749, 750,
133, 146, 154, 159, 850, 779, 626, 588, 865, 864, 851, 846, 791, 798, 801, 799,
145, 156, 173, 176, 549, 372, 335, 336, 847, 859, 845, 841, 800, 789, 763, 716,
179, 186, 190, 193, 322, 321, 291, 275, 827, 818, 811, 832, 706, 672, 679, 664,
185, 187, 182, 174, 277, 280, 279, 263, 814, 813, 812, 802, 665, 653, 650, 660,
165, 161, 143, 147, 258, 262, 267, 257, 803, 783, 787, 765, 652, 651, 654, 644,
130, 109, 105, 108, 255, 252, 234, 225, 723, 742, 747, 753, 639, 643, 671, 678,
104, 100, 91, 86, 211, 208, 206, 205, 766, 757, 758, 770, 680, 667, 655, 641,
82, 73, 68, 69, 74, 202, 194, 196, 207, 776, 774, 768, 756, 640, 804, 826, 830,
66, 63, 60, 54, 52, 215, 217, 221, 192, 762, 767, 760, 771, 834, 835, 645, 628,

627, 596, 587, 573, 461, 440, 435, 428, 419, 378, 376, 352, 884, 887, 897, 871,
562, 500, 493, 450, 427, 418, 417, 406, 375, 341, 313, 292, 873, 877, 878, 817,
449, 432, 379, 373, 400, 401, 402, 407, 283, 259, 253, 250, 823, 558, 284, 200,
370, 367, 366, 342, 413, 420, 421, 415, 268, 282, 333, 326, 99, 89, 77, 80, 274,
339, 344, 340, 331, 414, 403, 404, 391, 305, 324, 244, 487, 7, 1, 5, 9, 10, 4, 3, 2,
337, 363, 360, 397, 384, 382, 386, 388, 477, 513, 517, 525, 15, 34, 0
464, 610, 528, 526, 389, 387, 395, 399, 503, 725, 793, 810,

Distancia total: 113926

Apéndice 6: Resultado de la Ruta Menos Costosa con el Enfoque Híbrido para Zimbabwe

Recorrido optimizado:

0, 5, 1, 7, 39, 62, 43, 463, 480, 492, 491, 513, 477, 487, 469, 725, 793, 810, 743,
31, 24, 23, 22, 25, 504, 503, 525, 517, 468, 457, 475, 485, 685, 726, 777, 794,
16, 11, 6, 8, 18, 26, 497, 489, 490, 479, 476, 486, 502, 512, 686, 663, 634, 609,
51, 65, 67, 78, 131, 459, 460, 470, 461, 516, 511, 510, 509, 595, 590, 586, 580,
265, 271, 281, 264, 440, 435, 428, 427, 496, 474, 467, 451, 579, 585, 603, 608,
260, 269, 285, 270, 418, 417, 406, 400, 466, 473, 484, 495, 607, 598, 589, 594,
284, 200, 99, 89, 401, 402, 407, 413, 515, 521, 522, 524, 593, 584, 576, 571,
274, 80, 77, 41, 40, 420, 421, 415, 414, 523, 530, 535, 538, 555, 546, 536, 531,
21, 45, 48, 81, 85, 403, 404, 391, 384, 544, 543, 537, 534, 556, 568, 569, 577,
84, 83, 87, 93, 92, 381, 385, 392, 393, 508, 550, 559, 563, 572, 557, 547, 540,
236, 254, 296, 282, 394, 396, 405, 398, 564, 565, 551, 545, 526, 505, 498, 499,
268, 250, 253, 259, 388, 386, 382, 377, 539, 552, 553, 560, 506, 518, 527, 532,
283, 292, 313, 341, 357, 355, 348, 345, 554, 561, 566, 567, 541, 528, 610, 493,
375, 352, 376, 378, 346, 333, 327, 343, 583, 600, 604, 582, 450, 449, 432, 379,
389, 387, 395, 399, 326, 305, 324, 244, 581, 599, 617, 618, 373, 370, 367, 366,
419, 410, 408, 409, 438, 443, 433, 426, 622, 614, 623, 624, 342, 339, 344, 340,
416, 422, 423, 431, 439, 434, 444, 445, 605, 601, 602, 606, 331, 337, 363, 360,
430, 429, 441, 454, 456, 452, 446, 447, 611, 619, 620, 615, 318, 314, 329, 325,
462, 448, 455, 471, 453, 458, 478, 488, 633, 635, 662, 661, 315, 309, 303, 304,

297, 306, 293, 272, 740, 752, 748, 737, 832, 811, 818, 827, 321, 322, 336, 335,
273, 256, 261, 251, 734, 728, 715, 713, 841, 845, 859, 847, 372, 549, 588, 626,
132, 102, 96, 88, 72, 697, 702, 683, 658, 846, 851, 838, 844, 779, 824, 840, 829,
59, 46, 36, 32, 27, 642, 677, 668, 670, 858, 868, 875, 894, 819, 821, 822, 745,
37, 38, 44, 50, 54, 666, 674, 657, 656, 899, 864, 865, 872, 597, 533, 520, 519,
52, 53, 55, 60, 63, 659, 669, 673, 676, 883, 878, 877, 873, 507, 383, 358, 354,
68, 73, 82, 95, 98, 681, 684, 694, 693, 871, 890, 893, 898, 365, 362, 359, 347,
229, 266, 286, 289, 699, 711, 718, 724, 903, 904, 906, 911, 338, 332, 330, 307,
278, 288, 299, 298, 701, 689, 688, 696, 909, 910, 912, 915, 308, 287, 178, 56,
302, 294, 310, 316, 698, 703, 710, 720, 917, 916, 907, 913, 57, 49, 79, 76, 71,
320, 356, 361, 353, 712, 727, 731, 739, 918, 920, 922, 927, 64, 34, 15, 47, 42,
351, 350, 368, 371, 735, 733, 746, 751, 925, 924, 923, 921, 75, 94, 90, 97, 101,
380, 390, 411, 437, 738, 730, 722, 714, 926, 928, 919, 914, 204, 153, 237, 203,
424, 578, 612, 637, 705, 717, 709, 695, 908, 905, 901, 900, 180, 187, 214, 199,
647, 648, 754, 816, 671, 678, 680, 667, 896, 895, 902, 867, 222, 225, 211, 208,
744, 839, 848, 852, 655, 651, 654, 644, 860, 850, 828, 831, 206, 221, 217, 215,
820, 646, 636, 630, 639, 643, 652, 660, 836, 837, 842, 862, 207, 196, 194, 202,
687, 823, 817, 729, 650, 653, 665, 691, 876, 882, 886, 879, 205, 192, 184, 181,
795, 629, 591, 592, 682, 675, 690, 692, 880, 874, 885, 889, 172, 152, 158, 149,
574, 570, 613, 632, 700, 708, 719, 707, 891, 892, 888, 869, 142, 135, 134, 136,
616, 558, 514, 472, 704, 721, 732, 741, 866, 856, 849, 855, 139, 137, 141, 148,
482, 481, 501, 442, 736, 755, 764, 749, 833, 825, 796, 759, 151, 144, 160, 163,
436, 397, 464, 500, 750, 767, 760, 771, 638, 625, 631, 649, 167, 169, 189, 201,
562, 573, 587, 596, 782, 785, 784, 791, 621, 575, 425, 483, 216, 228, 234, 235,
627, 628, 645, 723, 798, 801, 799, 800, 548, 542, 529, 494, 226, 230, 224, 232,
765, 787, 783, 814, 789, 769, 780, 773, 465, 412, 369, 334, 241, 238, 239, 240,
813, 812, 802, 803, 778, 781, 772, 761, 323, 364, 374, 349, 242, 246, 247, 249,
788, 792, 790, 815, 763, 716, 706, 664, 319, 317, 328, 312, 245, 248, 243, 223,
807, 805, 806, 809, 679, 672, 641, 640, 311, 295, 301, 300, 209, 212, 218, 220,
808, 797, 786, 775, 804, 826, 830, 843, 290, 276, 267, 257, 231, 219, 191, 170,
762, 756, 768, 774, 853, 863, 884, 887, 255, 252, 227, 233, 150, 112, 111, 106,
776, 770, 758, 757, 897, 881, 870, 861, 262, 258, 263, 279, 107, 121, 197, 183,
766, 753, 747, 742, 857, 854, 834, 835, 280, 277, 275, 291, 195, 210, 213, 198,

188, 177, 175, 171, 125, 127, 122, 115, 190, 193, 185, 182, 58, 61, 30, 13, 17,
168, 162, 157, 155, 110, 117, 118, 126, 174, 165, 161, 143, 29, 19, 28, 35, 33,
140, 166, 164, 138, 124, 120, 133, 146, 147, 130, 109, 105, 20, 14, 12, 9, 10, 4,
128, 116, 129, 113, 154, 159, 145, 156, 108, 104, 100, 91, 3, 2, 0
123, 114, 119, 103, 173, 176, 179, 186, 86, 69, 74, 66, 70,

Distancia optimizada: 103996