Aplicación informática KPTS (Kruskal, Prim, Tabu Search)

Julia Argentina Granera[1]
Víctor Manuel Valdivia[2]
María Elena Blandón Dávila[3]

RESUMEN

En este artículo se muestra la aplicación de una herramienta informática basada en teoría de grafos para analizar y resolver problemas de las rutas más cortas, utilizando los algoritmos de Prim, Kruskal y de búsqueda local de Tabú Search. Para el desarrollo de esta aplicación se utilizaron los siguientes elementos: Visual Studio 2010, librería GraphSharp y librería QuickGraph. Para la creación de esta herramienta, se estableció una estructura de clases que diera soporte a los gráficos: 1) PocGraph: representa el grafo; 2) PocEdge: representa las aristas del grafo; y 3) PocVertex: representa los nodos o vértices del grafo. Tanto el método de Kruskal como Prim generan un árbol mínimo recubridor del grafo, el cual consiste en un subgrafo del original. El algoritmo de Prim se trabajó con el objetivo de encontrar el árbol recubridor más corto; mientras que el algoritmo de Kruskal, con la finalidad de hallar el árbol minimal a partir de instancias TSP. El método de Tabú Search se aplica para encontrar el mínimo camino cerrado que une todos los vértices o nodos. Se diseñó el algoritmo de Tabú Search para minimizar las rutas partiendo de una solución inicial la cual se va modificando hasta obtener el resultado.

Palabras claves: Algoritmo de Prim, Algoritmo de Kruskal, Algoritmo de Tabú Search, Herramienta informática.

Recibido: 15 de febrero de 2016
Aceptado: 3 de abril de 2016

1 Universidad Nacional Autónoma de Nicaragua, Managua. Correo Electrónico: juliagranera@yahoo.es
2 Universidad Nacional Autónoma de Nicaragua, Managua. Correo Electrónico: victorvaldivia584@hotmail.com
3 Universidad Nacional Autónoma de Nicaragua, Managua. Correo Electrónico: blandondavila@gmail.com

KPTS computer application (Kruskal , Prim , Tabu Search)

ABSTRACT

This article describes the application of a software tool based on graph theory to analyze and solve problems of the shortest routes, using the algorithms of Prim, Kruskal and Tabu. For the development of this application the following elements were used: Visual Studio 2010, GraphSharp and QuickGraph. TO create this tool, a class structure that would support the graphics was established:1) PocGraph : represents the graph ; 2) PocEdge: represents the edges of the graph ; and 3) PocVertex : represents the nodes or vertices of the graph. The Prim algorithm worked with the aim of finding the shortest spanning tree; while Kruskal's algorithm , in order to find the minimal tree from TSP instances . Tabu Search method is applied to find the minimum closed road connecting all the vertices or nodes. The Tabu Search algorithm was designed to minimize the routes from an initial solution which is modified to obtain the result.

Keywords: Prim algorithm, Kruskal algorithm, Tabu Search Algorithm, Computer tool .

INTRODUCCIÓN

El ser humano siempre ha tenido la necesidad de recorrer muchos lugares, para ello han utilizado caminos estratégicos y cortos, buscando hallar la ruta óptima con el mayor ahorro de tiempo, energía, distancia, entre otros, recorriendo todos los puntos designados. En la actualidad se puede apreciar que existen muchas cosas que pueden parecer de lo más habitual: caminos, líneas de comunicación telefónica, televisión por cable, líneas aéreas, circuitos eléctricos de nuestras casas, automóviles. Lo que no se piensa frecuentemente es que estos forman parte de algo, que en matemática se denomina grafos.

El estudio de los problemas de rutas mínimas surge a mediados del siglo pasado con la propuesta del modelo matemático, del problema de agente viajero (Traveling Salesman Problem, TSP), a partir del cual se han realizado muchas investigaciones particulares y aplicadas al mundo real. En los últimos años, ha aumentado la implementación de herramientas informáticas para la resolución de problemas reales.

En este trabajo se aplicó la Teoría de Grafos para la optimización de rutas utilizando el Algoritmo de Prim, Kruskal y Tabú Search, con el propósito de comparar los resultados obtenidos, ejecutando dichos algoritmos con diferentes instancias.

De acuerdo a los resultados obtenidos, se evidencia, que no existen diferencias significativas para encontrar las rutas más cortas utilizando los métodos de Prim o Kruskal; sin embargo, en el caso del algoritmo de Tabú Search los valores proporcionados por esta aplicación se aproximan a los casos de TSP que se encuentran en el directorio TSP (de 0-1.3%)

DESARROLLO

Para el desarrollo de esta aplicación se utilizó Visual Studio 2010 por su facilidad del lenguaje, permite crea aplicaciones en muy poco tiempo; es decir, un desarrollo eficaz y de menor inversión tanto en tiempo como en dinero. Genera librerías, dinámicas de forma activa, mediante una reconfiguración en su proceso de colección o codificación.

Además, Visual Studio utiliza un lenguaje simple, por lo que es fácil de aprender, se dibujan formularios mediante el arrastre de controles. La sintaxis tiene semejanza al lenguaje natural humano, es un lenguaje compatible con Microsoft office, con el fin de conseguir en el menor tiempo posible los resultados que se desean obtener.

Este tiene una ligera implementación de la POO (Programación Orientada a Objetos), la cual es un paradigma de programación que usa objetos y sus interacciones para diseñar aplicaciones y programas de computadora. Por otro lado, es excelente para cálculos intensivos del CPU (Romero, 2012).

Así mismo, se aplicó Quick Graph[1] que es una aplicación gratuita que permite realizar gráficas de funciones en 2D y 3D, con la posibilidad de graficar ecuaciones, tanto en coordenadas polares, cartesianas como esféricas y cilíndricas. La visualización gráfica de las ecuaciones puede ser en malla o sólido.

Para el desarrollo de la aplicación se creó una estructura de clases que diera soporte a los gráficos. Para esto se utilizaron 3 clases:

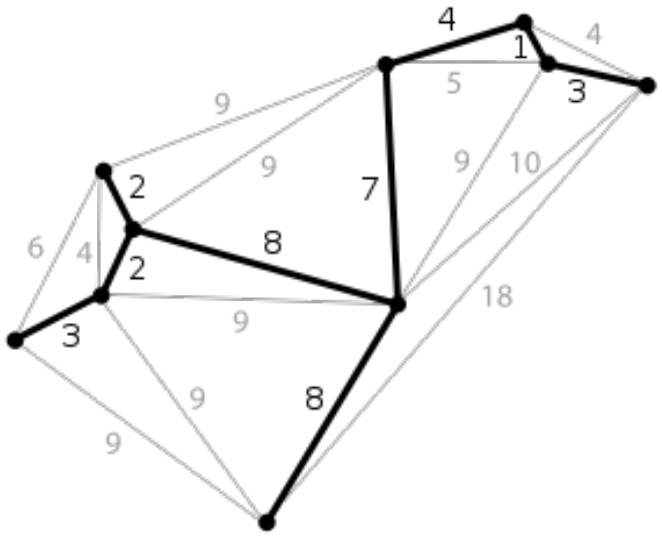
1. **PocGraph:** representa al grafo y está conformado por dos clases más. Esta clase deriva de BidirectionalGraph (Of PocVertex, PocEdge). Además, tiene un atributo, llamado peso.
2. **PocEdge:** representa las aristas del grafo, este deriva de Edge (Of PocVertex). Tiene dos atributos: ID que representa el nombre y peso de tipo doble.
3. **PocVertex:** representa los nodos o vértices del

1 Librería QuickGraph se puede encontrar en: https://quickgraph.codeplex.com y la librería GraphSharp que se puede encontrar en https://graphsharp.codeplex.com/

grafo. Tiene los siguientes atributos: ID que lo identifica, rank la posición que ocupa dentro del grafo y un apuntador root de tipo PocVertex, que puede apuntar a los hijos de dicho vértice.

Tanto el método de Kruskal como Prim generan un árbol mínimo recubridor del grafo y consiste en un subgrafo del original. A continuación, se muestran los seudocódigos utilizados para resolver dicha situación.

Árbol recubridor mínimo o MST (minimum-spanning-tree)



Un ejemplo de árbol recubridor mínimo: Cada punto representa un vértice, cada arista está etiquetada con su peso, que en este caso equivale a su longitud.

Dado un grafo conexo y no dirigido, un árbol recubridor mínimo de ese grafo es un subgrafo, que tiene que ser un árbol y contener todos los vértices del grafo inicial. Cada arista tiene asignado un peso proporcional entre ellos, que es un número representativo de algún objeto, distancia, entre otros. Se usa para asignar un peso total al árbol recubridor mínimo, computando la suma de todos los pesos de las aristas del árbol en cuestión.

Un árbol recubridor mínimo o un árbol expandido mínimo es un árbol recubridor, que pesa menos o igual que otros árboles recubridores. Todo grafo tiene un bosque recubridor mínimo.

Algoritmo de PRIM

Este algoritmo incrementa continuamente el tamaño de un árbol, se comienza por un vértice inicial al que se le van agregando sucesivamente vértices, cuya distancia a las anteriores es mínima. La idea básica de este algoritmo consiste en añadir en cada paso, una arista de peso mínimo a un árbol previamente construido. Más explícitamente:

Paso 1: Se elige un vértice u de G y se considera el árbol S={u}
Paso 2: Se considera la arista e de mínimo peso que une un vértice de S y un vértice que es de S, y se hace S =S+e
Paso 3: Si el número de aristas de T es n -1 el algoritmo termina. En caso contrario se vuelve al paso 2. (Cárdenas, Perez Torrez , & Barrera).

Algoritmo de Kruskal

El algoritmo de Kruskal resuelve la misma clase de problema que el de Prim, salvo que en esta ocasión no se parte desde ningún nodo elegido al azar para resolver el mismo problema; lo que se hace es pasarle a la función, una lista con las aristas ordenadas de menor a mayor y se va tomando una para formar el árbol recubridor mínimo. Hay que seguir los siguientes pasos (Cárdenas, Perez Torrez , & Barrera):

- Se marca la arista con menor valor. Si hay más de una, se elige cualquiera de ellas.
- De las aristas restantes, se marca la que tenga menor valor, si hay más de una, se elige cualquiera de ellas.
- Repetir el paso 2 siempre que la arista elegida no forme un ciclo con las ya marcadas.
- El proceso termina cuando se tienen todos los nodos del grafo en alguna de las aristas marcadas,

es decir, cuando se tienen marcados n – 1 arcos, siendo n el número de nodos del grafo.

A continuación se muestran los seudocódigos de los algoritmos

Function Kruskal(G)
Para cada v en V[G] hacer
Nuevo conjunto C(v) ← {v}.
Nuevo heap Q que contiene todas las aristas de G, ordenando por su peso.
Defino un árbol T ← Ø
// n es el número total de vértices
Mientras T tenga menos de n-1 vértices hacer
(u,v) ← Q.sacarMin()
// previene ciclos en T. agrega (u,v) si u y v están diferentes componentes en el conjunto.
// Nótese que C(u) devuelve la componente a la que pertenece u.
if C(v) ≠ C(u) then
Agregar arista (v,u) a T.
Merge C(v) y C(u) en el conjunto
Return árbol T

Prim
Prim (Grafo G)
/* Inicializamos todos los nodos del grafo.
La distancia la ponemos a infinito y el padre de cada nodo a NULL
Encolamos, en una cola de prioridad, donde la prioridad es la distancia, todas las parejas <nodo,distancia> del grafo*/
por cada u en V[G] hacer
distancia[u] = INFINITO
padre[u] = NULL
Añadir(cola,<u,distancia[u]>)
distancia[u]=0
mientras !está vacía(cola) hacer
// OJO: Se entiende por mayor prioridad aquel nodo cuya distancia[u] es menor.
u = extraer mínimo (cola) //devuelve el mínimo y lo elimina de la cola.
por cada v adyacente a 'u' hacer
si ((v □ cola) && (distancia[v] > peso(u, v)) entonces
padre[v] = u
distancia[v] = peso(u, v)
Actualizar(cola,<v,distancia[v]>)

A partir de estos algoritmos se inició la codificación de las funciones, que encuentran los árboles generadores mínimos, quedando las mismas de la siguiente forma:

```
Kruskal
For Each ed In arreglo
   root1 = GetRoot(ed.Source)
   root2 = GetRoot(ed.Target)
   If root1.ID <> root2.ID Then
      totalCost += ed._peso
      Join(root1, root2)
      AddNewGraphEdge(ed.Source, ed.Target, ed._
      peso)
      contador = contador + 1
    End If
Next
MsgBox("Costo Total: " & totalCost, , "Método
Kruskal")
```

Prim

```
For i = 0 To v
   key(i) = 10000
   mstSet(i) = False
Next
key(0) = 0
parent(0) = -1
For count = 0 To v - 2
     u = minKey(key, mstSet, v1)
     mstSet(u) = True
     For v1 = 0 To v - 1
          If (graph(u, v1) <> 0 And mstSet(v1) = False)
          And (graph(u, v1) < key(v1)) Then
          parent(v1) = u
          key(v1) = graph(u, v1)
          End If
    Next
Next
```

```
Dim totalCost As Integer = 0
For i = 1 To prim.VertexCount - 1
    AddNewGraphEdgePrim(lista
    Vértices(parent(i)), listaVértices(i), graph(i,
    parent(i)))
    totalCost = graph(i, parent(i)) + totalCost
Next
prim._peso = totalCost
MsgBox("Costo Total: " & totalCost, , "Método Prim")
```

El caso de la búsqueda Tabú es completamente distinto, puesto que, genera un camino cerrado, donde se recorren cada uno de los nodos o vértices del grafo.

Problema del viajante

El Problema del Agente Viajero (TSP por sus siglas en inglés) o problema del viajante, responde a la siguiente pregunta: Dada una lista de ciudades y las distancias entre cada par de ellas, ¿Cuál es la ruta más corta posible que visita cada ciudad exactamente una vez y regresa a la ciudad origen?

El problema fue formulado por primera vez en 1930 y es uno de los problemas de optimización más estudiados. Es usado como prueba para muchos métodos de optimización.

Aunque el problema es computacionalmente complejo, una gran cantidad de heurísticas y métodos exactos son conocidos, de manera que, algunas instancias desde cien hasta miles de ciudades pueden ser resueltas.

El TSP tiene diversas aplicaciones, aún en su formulación más simple, tales como: la planificación, la logística y en la fabricación de microchips. Un poco modificado, aparece como un sub-problema en muchas áreas, como en la secuencia de ADN.

En la teoría de la complejidad computacional, la versión de decisión del TSP (donde, dado un largo "L", la tarea es decidir cuál grafo tiene un camino menor que L) pertenece a la clase de los problemas NP-completos. Por tanto, es probable que en el caso peor, el tiempo de ejecución para cualquier algoritmo que resuelva el TSP aumente de forma exponencial con respecto al número de ciudades (Applegate, Bixby, Chvátal, & Cook, 2006).

A continuación se presenta el algoritmo para búsqueda Tabú (Tabú Search):

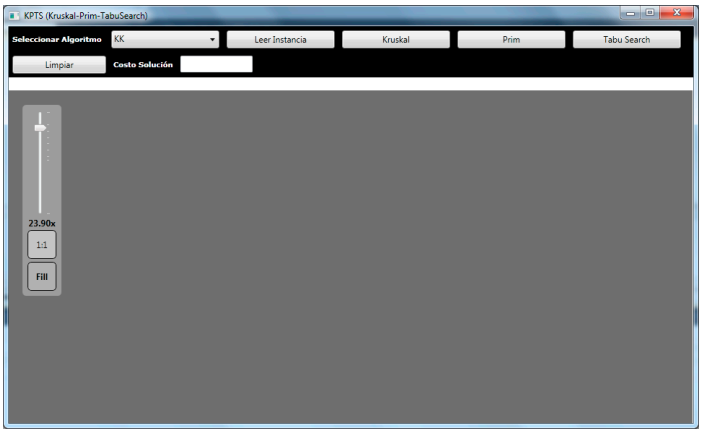
Algoritmo Tabú Search
Paso 0: Inicialización.
    X := solución inicial factible
    tmax := máximo número de iteraciones
    Mejor solución:= X
    Número de soluciones = t := 0
    Lista tabú:= vacía
Paso 1: Parada
    Si cualquier movimiento posible de la solución actual es tabú o si
    t=tmax
    Entonces parar. Entregar Mejor solución.
Paso 2: Mover
    Elegir el mejor movimiento no-tabú factible x(t+1)
Paso 3: Iteración.
    Modificar X(t+1) := X(t) + x(t+1)
Paso 4: Reemplazar el mejor.
    Si el valor de la función objetivo de X(t+1) es superior a Mejor solución entonces a Mejor solución := X(t+1)
Paso 5: Actualizar Lista Tabú.
    Eliminar desde la lista tabú cualquier movimiento que ha permanecido un suficiente número de iteraciones en la lista.
    Agregar un conjunto de movimientos que involucran un retorno inmediato
      Desde X(t+1) a X(t)
Paso 6: Incrementar
    t := t+1, Volver a Paso 1.

A continuación se muestra la interfaz principal de la herramienta informática que se desarrolló con la ayuda de Visual Studio 2010

Interfaz Principal

En la pantalla principal de KPTS, se pueden encontrar los siguientes elementos:

- **Una lista desplegable:** Contiene los algoritmos para el ordenamiento de grafos.
- **El botón Leer Instancia:** Selecciona un archivo de texto, que contiene la matriz de pesos del grafo.
- **El botón Kruskal:** Aplica el método de Kruskal para encontrar un árbol mínimo.
- **El botón Prim:** Aplica el método de Prim para encontrar un árbol mínimo.
- **El botón Tabú Search:** aplica el método de Tabú Search para encontrar el mínimo camino cerrado que une todos los vértices o nodos.
- **El botón Limpiar:** elimina el árbol actual y actualiza el costo de la solución.
- **Una caja de texto:** donde se muestra el costo generado, según el método aplicado.
- **Un control deslizante:** para hacer zoom en el grafo con la opción de Fill (ajustar al tamaño de pantalla) o según el usuario realice el ajusta.

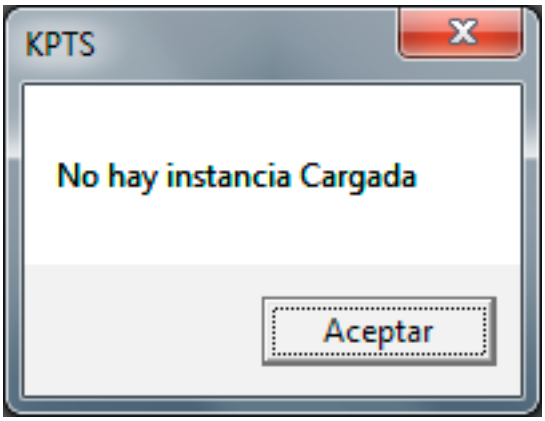


Ruta para la evaluación

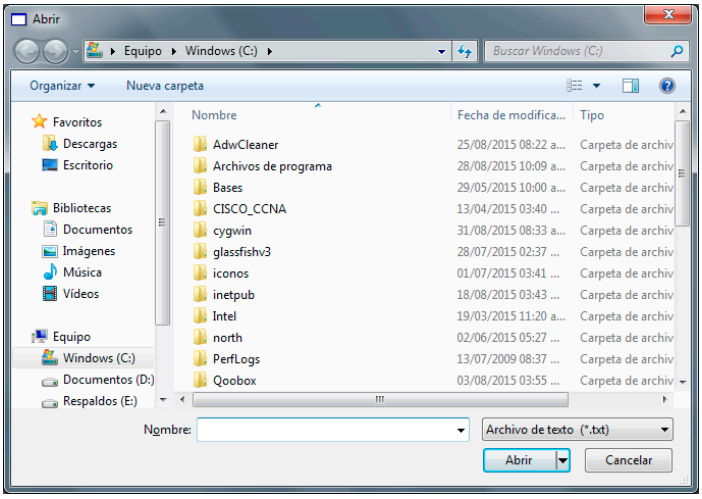
1. Cargar una instancia.
2. Aplicar cualquiera de los métodos: Kruskal, Prim o Tabú Search.

Opción de cargar una instancia

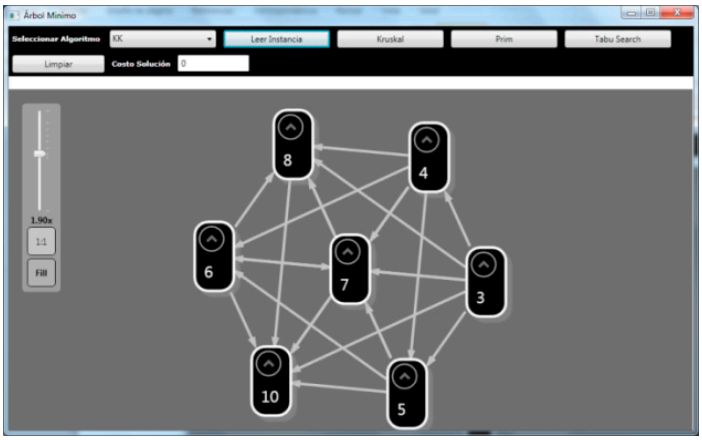
Para aplicar cualquiera de los métodos se hace necesaria una instancia a evaluar, para esto lo primero que debemos hacer es cargar cualquiera de las mismas. En caso de no hacerlo, se recibirá un mensaje, indicando que no existe instancia cargada.



Al hacer clic en Leer Instancia, se mostrará la siguiente ventana, donde debemos seleccionar la instancia a graficar.



Una vez encontrado el archivo se da clic en abrir y la aplicación mostrará el siguiente resultado.



En este momento puede aplicar los distintos métodos de ordenamiento del grafo, los disponibles son:

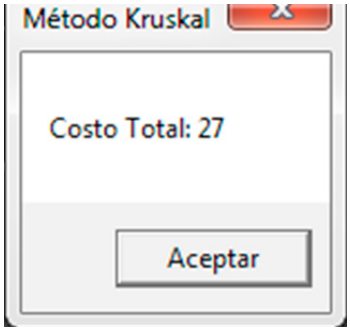
1. BoundedFR (variante circular de Fruchterman y Reingold)
2. Circular (círculo)
3. CompoundFDP
4. FR (Fruchterman y Reingold)
5. ISOM (Isométrico)
6. KK (Kamada y Kawai)
7. LinLog (fuerzas atractivas, fuerzas repulsivas)
8. Tree (árbol)

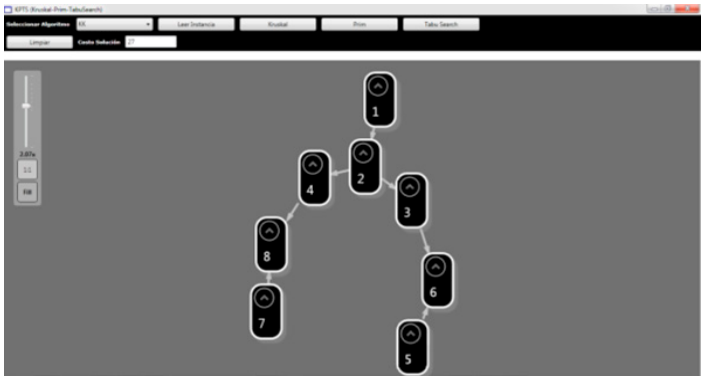
Por defecto aparece seleccionado el Algoritmo KK.

Una vez cargada la instancia a evaluar, se procede a seleccionar el algoritmo a aplicar, haciendo clic en el botón correspondiente. A modo de ejemplo, se utilizó la instancia tsp8. A continuación, se muestran resultados.

Kruskal

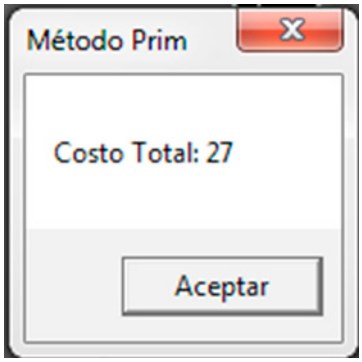
Al ejecutar el método de Kruskal, se mostrará tanto el peso por medio de una ventana de mensaje como en la pantalla principal.

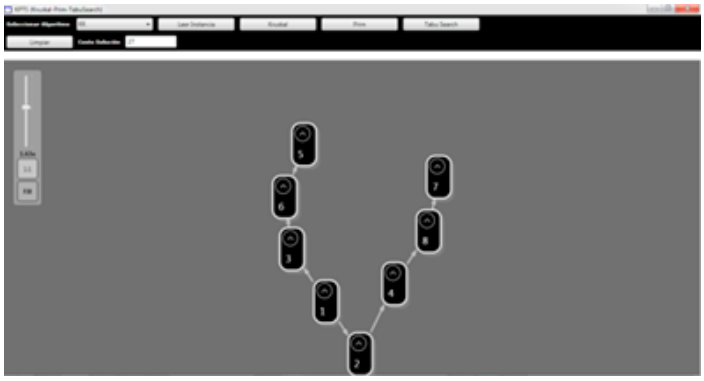




Prim

De igual forma, si se pulsa el botón de Prim, se mostrará el mensaje de texto con el peso total y graficará el correspondiente MST.

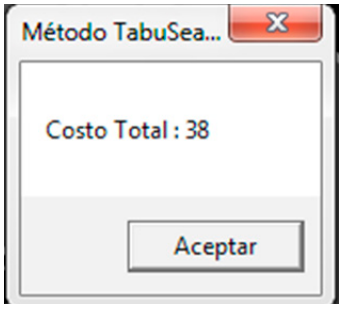


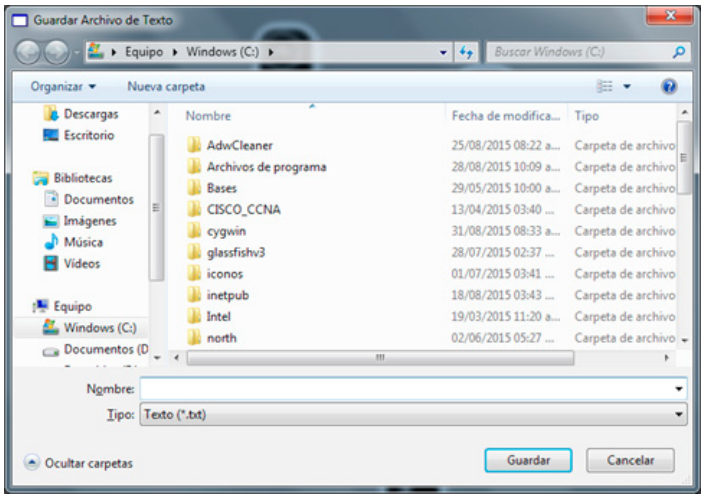


Tabu Search

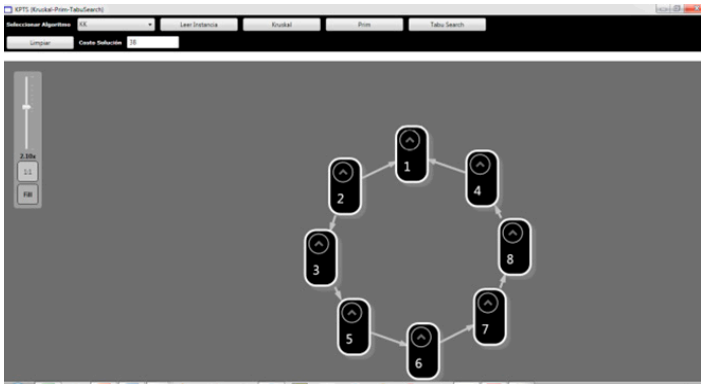
Para el caso de Tabu Search, este generará un camino cerrado, que contiene el mínimo recorrido que une todos los nodos o aristas, en este caso nos referimos a un TSP (Travel Salesman Path).

Al pulsar el botón de Tabú Search, se mostrará una ventana, solicitando donde se guardará el archivo de salida, que corresponde a los distintos cambios ocurridos durante cada una de las iteraciones que se realizan (10000).



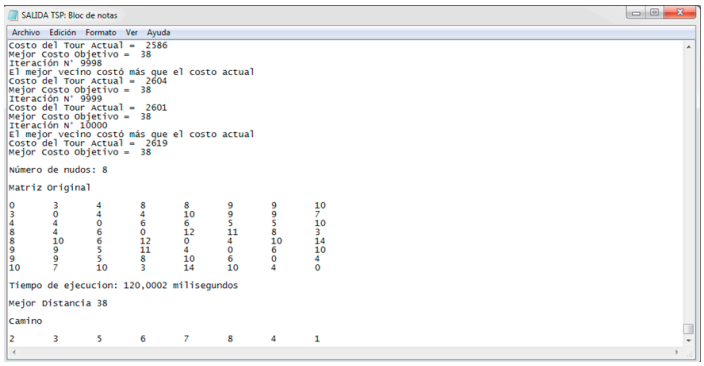


El resultado será una ventana, que muestra, el peso del camino generado y el correspondiente gráfico.



La aplicación muestra la salida en formato .txt.

A continuación, se presenta la última pantalla de este resultado.



El botón de Limpiar

Eliminará el grafo actual y pondrá el costo de la solución a 0.

Para ejecutar los métodos es necesario cargar la instancia correspondiente.

De acuerdo a los resultados obtenidos se evidencia, que no existen diferencias significativas para encontrar las rutas más cortas utilizando los métodos de Prim o Kruskal. En el caso del algoritmo de Tabú Search, los valores proporcionados por esta aplicación se aproximan a los casos de TSP, que se encuentran en el directorio TSP (de 0 -1.3%)

El código fuente de la herramienta informática se muestra a continuación:

```
Class MainWindow
    Inherits Window
    Private vm As MainWindowViewModel
    Public Sub New()
        vm = New MainWindowViewModel()
        Me.DataContext = vm
        ' Llamada necesaria para el diseñador.
        InitializeComponent()
        ' Agregue cualquier inicialización después
de la llamada a InitializeComponent().
    End Sub
    Private Sub Button_Click(ByVal sender As Ob-
ject, ByVal e As RoutedEventArgs)
        vm.ReLayoutGraph()
    End Sub
    Private Sub Leer_Instancias(ByVal sender As
Object, ByVal e As RoutedEventArgs)
        ' vm.ReLayoutGraph()
        vm.CrearInstancia()
    End Sub
    Private Sub Kruskalizador(ByVal sender As Ob-
ject, ByVal e As RoutedEventArgs)
        ' vm.ReLayoutGraph()
        vm.Kruskalizar()
    End Sub
    Private Sub Primalizar(ByVal sender As Object,
ByVal e As RoutedEventArgs)
        ' vm.ReLayoutGraph()
```

```
        vm.Primalizar()
    End Sub
    Private Sub TabuS(ByVal sender As Object, By-
Val e As RoutedEventArgs)
        ' vm.ReLayoutGraph()
        vm.TabuSearch()
    End Sub
End Class
```

CONCLUSIONES

La aplicación desarrollada para encontrar las rutas más cortas con diferentes instancias, permitió demostrar la importancia de la implementación de herramientas informáticas para resolver problemas, con el fin de optimizar costos, tiempo y recursos.

La aplicación desarrollada generó nuevos conocimientos, valorando la interrelación de la programación matemática con las herramientas informáticas para la obtención de una solución óptima a problemas de la vida cotidiana.

La herramienta que se diseñó permite el llamado de cualquier instancia (archivo .txt en forma matricial). Esta herramienta es útil, con la creación de nuevas instancias, para la resolución de problemas reales como ruta de distribución, cableado, transporte terrestre, entre otros.

A partir del uso de seudocódigos ya existentes, se llegó a crear la aplicación de los algoritmos, siendo muy útil para cálculos de rutas más cortas.

AGRADECIMIENTOS

Se le agradece el apoyo al MSc. y Ph.D. Ramón Antonio Parajón Guevara docente investigador de la UNAN –Managua, por los aportes brindados para la realización de esta aplicación.

BIBLIOGRAFÍA DE REFERENCIA

Applegate, D., Bixby, R., Chvátal, V., & Cook, W. (2006). The Traveling Salesman Problem. Michigan , USA: Princeton University Press.

Capuano, B. (2010). Visual Studio 2010. NET 4.0. Madrid, España: krasis Press.

Cárdenas, J. R., Pérez Torrez , F., & Barrera, E. (s.f.). http://es.slideshare.net. Recuperado el 14 de Marzo de 2016, de http://es.slideshare.net: http://es.slideshare.net/fher969/algoritmos-de-kruskal-y-prim

Correa Espinal, A., Cogollo Flórez, J., & Salazar López, J. (Diciembre de 2011). Solución de problemas de ruteo de vehículos con restricciones de capacidad usando la teoría de grafos. Avances en Sistemas e Informática , 27-32.

Graph#. (27 de Agosto de 2010). Recuperado el 20 de junio de 2015, de https://graphsharp.codeplex.com/

QuickGraph, Graph Data Structures and Algorithms for .NET. (19 de Noviembre de 2011). Recuperado el 20 de Junio de 2015, de https://quickgraph.codeplex.com/

Romero Cueto, B. E. (10 de Mayo de 2012). http://sites.google.com. Recuperado el 10 de Marzo de 2016, de http://sites.google.com: https://sites.google.com/site/evolucionvisualbasic/classroom-news/thisweekisscienceweek

Tabu Search. (s.f.). Recuperado el 21 de Junio de 2015, de http://www.inf.utfsm.cl/~mcriff/IA/tabu-search.html