# Variante Heurística para el Problema del Viajante. Caso de Aplicación: Circuito de Pesca Deportiva

Ana Priscila Martinez[1],
Lidia Marina López[2]

Universidad Nacional del Comahue, Facultad de Informática, Neuquén, Argentina.
[1] martinezanapriscila@gmail.com
[2] lidia.lopez@fi.uncoma.edu.ar

**Abstract.** El presente trabajo se enfoca en la construcción de un algoritmo para resolver un circuito de pesca deportiva, aplicando técnicas de optimización combinatoria con el objeto de generar la mejor solución al problema del recorrido para la pesca deportiva en la provincia de Neuquén. La planificación y gestión de caminos para recorridos con preferencias exige disponer de sistemas eficientes de optimización de rutas. Su complejidad es exponencial. Para la resolución de este tipo de problemas se deben emplear heurísticas que permitan soluciones factibles. Para modelar un circuito turístico asociado a la pesca deportiva se utiliza la exploración de un grafo con restricciones. El mismo se encuadra dentro del Problema del Viajante. Se propone el diseño de un algoritmo metaheurístico de búsqueda tabú, basado en una búsqueda local, para encontrar una solución al problema.

**Palabras Claves:** Grafo, Heurística, Problema del Viajante, Búsqueda Tabú, Algoritmo.

## 1 Introducción

El presente trabajo se enfoca en la construcción de un algoritmo para resolver un circuito turístico con determinadas características, aplicando técnicas de optimización combinatoria con el objeto de generar la mejor solución al problema del recorrido para la pesca de salmónidos en la provincia de Neuquén.

### 1.1 Objetivo

El objetivo principal de este trabajo es el estudio, construcción e implementación de un algoritmo que resuelva el armado de un circuito turístico específico, en este caso para la pesca deportiva, mediante técnicas de exploración de grafos. Es decir, dada una fecha de inicio y fin del recorrido, tipo de pesca y ciudad de origen, devuelva un camino óptimo con todos los accesos de pesca habilitados para esas especificaciones.

El diseño de recorridos óptimos o con restricciones sobre rutas geográficas involucra la aplicación de algoritmos de exploración de grafos y de sistemas de información geográfica. Dado que estos caminos no se modifican, el problema de diseño de rutas, con preferencias determinadas, se encuadra en las variaciones del clásico *Problema del Viajante.* Este tipo de problemas no es abordable con técnicas de resolución exactas, salvo para problemas muy pequeños, debiéndose emplear heurísticas para encontrar soluciones factibles.

Este trabajo presenta un algoritmo basado en grafos representados por matrices de adyacencia, y una heurística para la obtención del grafo resultado.

## 2 El Problema del Viajante (TSP)

*Un viajante quiere visitar n ciudades, una y sólo una vez cada una, empezando por una cualquiera de ellas y regresando al mismo lugar del que partió. Supongamos que conoce la distancia entre cualquier par de ciudades. ¿De qué forma debe hacer el recorrido si pretende minimizar la distancia total?* [6]

A este problema se lo conoce con el nombre de *problema del viajante* o TSP (Travelling Salesman Problem) y resulta ser uno de los más prominentes dentro del campo de la optimización combinatoria. Es un problema de complejidad *NP- difícil.* A pesar de su aparente sencillez aún no se ha conseguido encontrar un algoritmo que sea capaz de resolverlo en un tiempo razonable (polinómico). [2]

Matemáticamente el TSP corresponde al problema de hallar un *ciclo hamiltoniano* de mínima distancia en un grafo completo $G = (V, E)$ en donde $V = \{1, 2, ...,n\}$ es el conjunto de nodos y representan las ciudades, $E$ es el conjunto de ramas que denotan la conexión entre ellas (rutas) y $d : E \rightarrow R^+$ una función que a cada $(i, j) \in E$ le asigna la distancia $d_{ij}$ entre las ciudades $i$ y $j$.

## 3 Métodos Algoritmos Heurísticos

*"Un método heurístico es un procedimiento para resolver un problema de optimización bien definido mediante una aproximación intuitiva, en la que la estructura del problema se utiliza de forma inteligente para obtener una buena solución."* [8]

En la *optimización combinatoria* se plantea un problema de optimización como un par *(F, h)* donde $F$ es el conjunto de soluciones factibles de una instancia de un problema de optimización y $h: F \rightarrow R^+$ es la función costo. [7]

El objetivo es encontrar un $f_0 \in F$ tal que $h(f_0) \leq h(f)$ para todo $f \in F$. La base del método consiste en asignar a cada $t \in F$ un entorno de $t$, es decir, un subconjunto $N(t) \subseteq F$ que contiene a $t$ y a cuyos elementos los consideramos vecinos de $t$.

El *algoritmo de búsqueda local* consiste en partir de una solución inicial $t_0$, encontrar un $s \in N(t)$ con coste menor, para de esa forma disminuir $h(t)$, y repetir este procedimiento hasta llegar a un $t$ en cuyo entorno resulte imposible disminuir el coste. Si esto pasa, se está en presencia de un *mínimo local*. [1]

Una *heurística,* es una función matemática *h (f)* que sirve como una estimación del coste del camino más económico de un nodo dado hasta el nodo objetivo. [8]

No garantizan una solución óptima, pero si aproximadas al resultado óptimo. Un buen algoritmo heurístico debe ser e*ficiente*, b*ueno*, y *robusto*.

## 4 Planteamiento del TSP con Algoritmos Heurísticos

Para enunciar el TSP a través de métodos heurísticos hay distintos tipos de categorías de heurísticas. Sólo se estudiara:

**Mejora Iterativa.** La idea básica consiste en comenzar con una solución inicial e ir realizando modificaciones para mejorar su calidad. El objetivo de esta técnica consiste en explorar iterativamente el estado de soluciones para encontrar las soluciones óptimas. Se detallan algunas técnicas a continuación [5]:

*Intercambio par a par*: El intercambio par a par involucra en cada iteración la eliminación de dos aristas y el reemplazo de éstas, con dos aristas diferentes que reconecten los fragmentos creados por la eliminación de las aristas produciendo un camino nuevo más corto. Esto es un caso especial del método *k-opt*.

*Heurística k-opt o heurística Lin-Kernighan*: Toma un recorrido dado y elimina *k* aristas mutuamente disjuntas. Reconecta los fragmentos conformando el recorrido, sin dejar ningún subcamino. Esto, en efecto, simplifica las consideraciones del TSP convirtiéndolo en un problema más simple. Cada extremo de un fragmento tiene $2k-2$ posibilidades de ser conectado: del total de $2k$-extremos de fragmentos disponibles, los dos extremos del fragmento que se está considerando son descartados. La técnica *k-opt* es un caso especial del *V-opt* o técnica *Variable-opt*.

*Heurística V-opt*: Mientras el método *k-opt* elimina un número fijo *k* de aristas del recorrido original, el método *variable-opt* no fija el tamaño del conjunto de aristas eliminadas. En cambio, este método aumenta el conjunto a medida que el proceso de búsqueda continúa. Este método es considerado la heurística más poderosa para el problema.

## 5 Caso de Estudio: Circuito de Pesca Deportiva

La construcción de un circuito turístico de pesca en la provincia de Neuquén, es el caso de estudio seleccionado para el desarrollo de una solución algorítmica aplicando las técnicas precedentes para el TSP.

### 5.1. Planteamiento del problema

El TSP corresponde a un recorrido de un grafo donde el origen viene determinado.

Para resolver el problema, se decidió implementar un algoritmo heurístico, ya que está demostrado que estos algoritmos son muy eficientes encontrando soluciones cercanas a la óptima.

**Modelo.** El mapa M de la región a estudiar, donde se encuentran georreferenciados los accesos de pesca, se modelará como un grafo *G = (V, E)*. Dicho grafo cumplirá que:

El conjunto de *aristas* no dirigidas $E \subseteq (V \times V)$ representará los segmentos de rutas o caminos entre dos accesos de pesca. El conjunto de *vértices V* representará entonces los accesos de pesca. Estos tendrán asociadas sus respectivas coordenadas geográficas respecto de algún punto en el espacio.

**Requerimientos.** El objetivo principal busca partir desde una región determinada, definir solamente los puntos en los que hay *accesos de pesca de salmónidos* (según la temporada en la que ese acceso está habilitado para la pesca, y el tipo de pesca), y volver al punto de partida.

Esta etapa de modelado requiere en primer lugar un proceso de obtención de la información cartográfica digital asociada a cada una de las zonas de pesca a trabajar y el pre-procesamiento realizado sobre la misma. A tal efecto, los accesos de pesca y sus caminos serán modelados por un grafo, y sus aristas (caminos) tendrán un costo asociado (la distancia/kilometraje).

### 5.2 Requerimientos

Se confeccionó la Matriz de Adyacencia, base generadora del grafo que mostrará gráficamente toda la información de los accesos de pesca. Esta matriz tiene la información en kilometraje, de todas las distancias que hay entre los distintos puntos del mapa, siempre y cuando exista un camino físico que los una. La información fue proporcionada por la Subsecretaría de Turismo de Neuquén. También se cuenta con la información de temporada y tipo de pesca para cada acceso de pesca.

Una vez confeccionada la matriz de adyacencia, se realiza el grafo, el cual es un grafo no dirigido y etiquetado, como peso de cada arco está el kilometraje entre cada nodo, y como nodo, las regiones detalladas en las tablas antes mencionadas.

### 5.3 Búsqueda Tabú

Este algoritmo es atribuido a Fred Glover quien menciono en 1986 [4]:

*"Es mejor una mala decisión basada en información, que una buena decisión al azar, ya que, en un sistema que emplea memoria, una mala elección basada en una estrategia proporcionará claves útiles para continuar la búsqueda. Una buena elección fruto del azar no proporcionará ninguna información para posteriores acciones".*

La búsqueda tabú [4] es un algoritmo metaheurístico, éste se aplica, generalmente, a problemas que no tienen un algoritmo o heurística específica que dé una solución satisfactoria.

La estructura de memoria más importante usada para determinar las soluciones permitidas es la *lista tabú*. Esta lista es la que permite salir de óptimos locales para alcanzar un óptimo global. En su forma más simple, una lista tabú es una memoria de corto plazo que contiene las soluciones que fueron visitadas en el pasado reciente

(menos de *n* iteraciones atrás, donde *n* es el número de soluciones previas que van a ser almacenadas, *n* es llamado la *tenencia del tabú*). Los atributos seleccionados de las soluciones recientemente visitadas son denominados *tabú-activos*. Las posibles soluciones que contengan elementos *tabú-activos* son *tabú*.

Cuando sólo un atributo es marcado como tabú, por lo general resulta en que más de una solución, es marcada como tabú. Algunas de estas soluciones, que ahora deben ser evitadas, podrían ser de excelente calidad y no serían visitadas. Para mitigar este problema, se introducen los *criterios de aspiración*: estos pueden modificar el estado de tabú de una solución, por lo tanto incluyendo la antes excluida solución en el conjunto de soluciones permitidas. Un criterio de aspiración muy utilizado es admitir soluciones que no son mejores que la *mejor solución* conocida al momento.

## 6 Solución Algorítmica Alcanzada

Se opta por desarrollar e implementar un *algoritmo metaheurístico de búsqueda tabú* basado en una búsqueda local, para encontrar una solución al problema.

En la búsqueda de la solución se codifica un algoritmo con el método de búsqueda tabú: *mejor vecino* y luego se fueron realizando *mejoras iterativas*.

El algoritmo propuesto comienza a buscar la solución en base a una *solución posible*. Esto es, a partir de un arreglo inicial con una posible solución, cuyo nodo inicial y final coinciden.

Los datos iniciales son: cantidad de nodos, nombre de los nodos y descripción, distancias entre los nodos, tipo de pesca, fecha de comienzo y fin de temporada según el tipo de pesca y la región.

Las estructuras de datos iniciales son: Matriz de distancias: *distancia*, Matriz de fecha de inicio y fin de temporada: *fechaTemp*, Matriz de tipos de pesca: *tipoPesca*, Arreglo con la solución inicial: *posibleSol*.

Para llenar con el contenido inicial el arreglo *posibleSol* se realizó un algoritmo que busca en la matriz de temporada *fechaTemp,* y tipos de pesca, *tipoPesca,* los accesos de pesca (nodos) que cumplan con los criterios ingresados por el usuario, los cuales son: Ciudad en la cual quiere empezar su recorrido, Fecha de inicio y fin del recorrido planeado, Tipo de pesca que va a realizar, las cuales pueden ser: Spinning, Fly Cast, Trolling o Spinning Exclusivamente con mosca. El arreglo *posibleSol* es el que se debe ordenar.

### 6.1 Codificación y Resolución del TSP mediante el Algoritmo de Búsqueda Tabú.

En esta sección se muestra cómo se resuelve el *Problema del Viajante (TSP)* usando una variación heurística del Algoritmo de *Búsqueda Tabú*.

El algoritmo de *Búsqueda Tabú* [15] es una mejora sobre la búsqueda local básica, que intenta superar los problemas de búsqueda local al no quedar atrapado en un mínimo local. De esta manera permite contemplar más resultados y compararlos; se logra permitiendo la *aceptación de movimientos que no mejoran el resultado*. Esta

implementación también permite escapar de soluciones sub-óptimas, mediante el uso de una *Lista Tabú*.

La lista Tabú es una lista de movimientos posibles que podrían incorporarse a una solución. Estos movimientos son operaciones de *intercambio*. En el caso en que se encuentra una nueva mejor solución, ese movimiento se hace *tabú* para un cierto *número de iteraciones.* Entonces, se añade a la lista de tabúes con un cierto valor denominado *tenencia Tabú* (Longitud Tabú), la cual se decrementa con cada iteración. Cuando esta es cero, el movimiento puede ser realizado y aceptado.

Una lista de tabú se representa como una matriz, donde cada celda representa la *tenencia tabú* de una operación de intercambio, si se intercambian dos ciudades. Esto evita que se retome un movimiento antes de explorar un poco.

Para permitir un movimiento tabú, es necesario aplicar criterios que permitan seleccionar un movimiento tabú en base a ciertas restricciones. Por ejemplo, un movimiento permite una nueva solución global mejor, por lo tanto, es aceptado, y su tenencia tabú se renueva.

En resumen, la *búsqueda tabú* realiza los siguientes pasos:

1. Crear una solución inicial, *posible solución*, luego llamarla *mejor solución*.
2. Encontrar el mejor vecino, basado en la *posible solución* aplicando ciertas operaciones de intercambio.
3. Si se alcanza el mejor vecino realizando un movimiento no tabú, se acepta como la nueva *mejor solución*, sino, encontrar otro mejor vecino.
4. Si se alcanza el número máximo de iteraciones (o alguna otra condición de parada), ir al paso cinco, caso contrario, seguir iterando desde el paso dos.
5. La mejor solución global, es la mejor solución al final de las iteraciones.

Este método heurístico está diseñado para escapar de la optimalidad local; se basa en el manejo y uso de una colección de principios que sirven para resolver el problema de manera inteligente, esto es, haciendo uso de memoria flexible para involucrar dos procesos: -la adquisición y, -el mejoramiento de la información; así, al tener cierta historia de los caminos ya recorridos y de los óptimos encontrados, se puede evitar permanecer en las mismas regiones, y recorrer regiones nuevas para encontrar otras soluciones mejores. [3]

**Resolución y codificación.** El código empieza en la clase de nivel superior *o main.* Esta clase contiene el método principal que inicializa los datos del problema y realiza la búsqueda tabú.

El algoritmo trata de encontrar la mejor solución vecina en cada iteración que no implique un movimiento tabú. La solución siempre es aceptada.

Cada iteración comienza desde la solución que dio la iteración anterior. Durante las iteraciones se realiza un seguimiento de cada solución generada y se almacena. La mejor solución, es la que se logra cuando el número predefinido de iteraciones ha terminado. A la hora de implementar el algoritmo, surgió el tema de la cantidad de caminos inexistentes entre dos pares de ciudades o accesos de pesca. Para ello se aplica Dijkstra (que devuelve el camino más corto entre dos nodos distintos) en cada iteración, antes de cada llamado al *getMejorVecino*.

A continuación la codificación en java el algoritmo del mejor vecino.

```
public static int[] getMejorVecino(ListaTabu tabuList,
AmbienteTSP ambienteTSP,int[] solInicial) {
     int[] mejorSol = new int[solInicial.length];
     System.arraycopy(solInicial, 0, mejorSol, 0,
mejorSol.length);
     int mejorCosto = ambienteTSP.getValorFuncion(solInicial);
     int ciudad1, ciudad2 = 0;
     boolean primerVecino = true;
     for (int i = 1; i <mejorSol.length - 1; i++) {
        for (int j=2; j<mejorSol.length-1; j++){
           int[] newMejorSol = new int[mejorSol.length];
           System.arraycopy(mejorSol, 0, newMejorSol, 0,
newMejorSol.length);
            newMejorSol = intercambio(i,j, solInicial);
            int newMejorCosto =
ambienteTSP.getValorFuncion(newMejorSol);
            if ((newMejorCosto > mejorCosto ||
primerVecino) &&
tabuList.listaTabu[i][j] == 0) {
                  primerVecino = false;
                ciudad1 = i;
                  ciudad2= j;
                System.arraycopy(newMejorSol, 0, mejorSol, 0,
newMejorSol.length);
                mejorCosto = newMejorCosto;
             }
          }
     }
     if (ciudad1 != 0 ) {
          tabuList.decrementaTabu();
          tabuList.moverTabu(ciudad1, ciudad2);
     }
      return mejorSol;
 }
```

Permite un movimiento que no mejora

No es un movimiento tabú. Solo cuando la tenencia tabú es cero, el movimiento puede ser aceptado

La *tenencia tabú* se decrementa con cada iteración

Cuando un movimiento se hace tabú se añade a la lista de tabúes.

La solución algorítmica encontrada combina adaptaciones de los algoritmos heurísticos iterativos: intercambio par a par, Búsqueda Tabú y de Dijkstra. También utiliza arreglos auxiliares con información útil para construir el recorrido y una estructura de pila.

**Datos de Prueba.**

Fecha desde: 02/07/2017 - Fecha Hasta: 01/08/2017

Ciudad Origen: Neuquen (Nodo 28) - Tipo de Pesca: Spinning (Int 0)

En la Figura 1 se muestra el grafo final y los resultados de las distintas estructuras utilizadas. En rojo están los *nodosPaso,* estos son los nodos por los cuales sólo se utilizan para unir dos destinos, y en verde los *nodosPesca.* En cuanto a las aristas o caminos están representados de la misma forma que los nodos. Se puede observar a simple vista que el camino que une el nodo 28 con el 29 es recorrido dos veces. Esto se puede deber a dos cosas: 1) no existen dos caminos distintos que la unan, o 2) al ser caminos de paso, ya que el nodo 29 al ser visitado por primera vez

es un *nodoPesca,* pero al visitarse por segunda vez es *nodoPaso,* éste no controla que los nodos ya hayan sido visitados. Como se aclaró en el presente trabajo, los caminos que sirven para unir dos *nodosPesca* que no son abordables mediante otros *nodosPesca*, no se controla si ya fueron visitados o no.

*Camino Inicial:* {28 29 27 13 9 6 3 1 0 28}
*Camino antes de Dijkstra:* {28 29 27 13 9 6 3 1 0 28}
*Solución final:* {28 29 27 13 9 11 10 7 6 5 3 1 0 29 28}
*Costo:* 1253 km

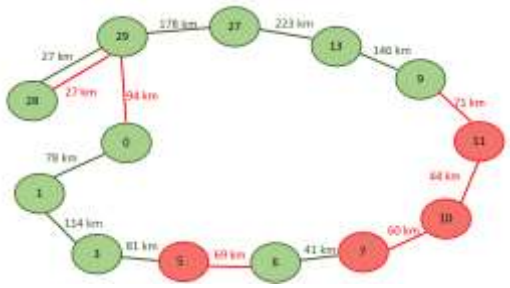

**Figura 1.** Grafo final del ejemplo uno. Distinguiendo *nodosPaso* de *nodosPesca.*

## 7 Eficiencia y Complejidad

Se modela un TSP perteneciente a problemas *NP-difíciles* debido a que el algoritmo solución conforma una transformación en tiempo polinomial del problema TSP genérico.

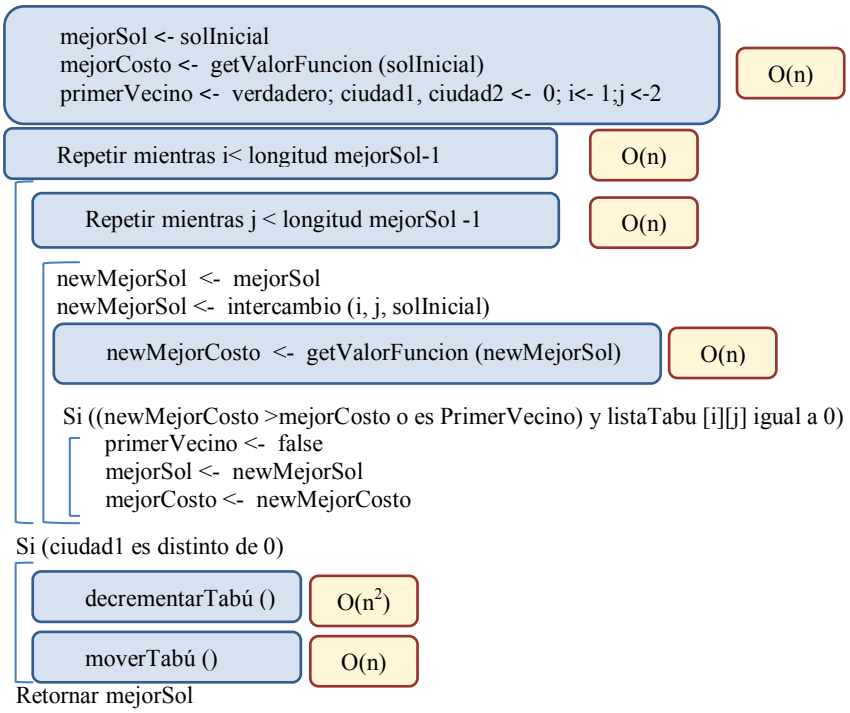


**Figura 2.** Diagrama de flujo y órdenes de ejecución del Algoritmo *GetMejorVecino* Final

En la Figura 2 se muestra el flujo de trabajo del algoritmo de mejor vecino junto con el orden de ejecución, el cual es O ($n^3$). En la Figura 3 se detalla un flujo general del algoritmo propuesto. Sobre el mismo se calcula el tiempo de ejecución total del algoritmo. Dicho algoritmo presenta un orden de ejecución polinomial, más precisamente $O(n^4)$.

El orden de un algoritmo se usa como medida de su eficiencia. Un algoritmo es considerado *eficiente* sólo en el caso polinomial.

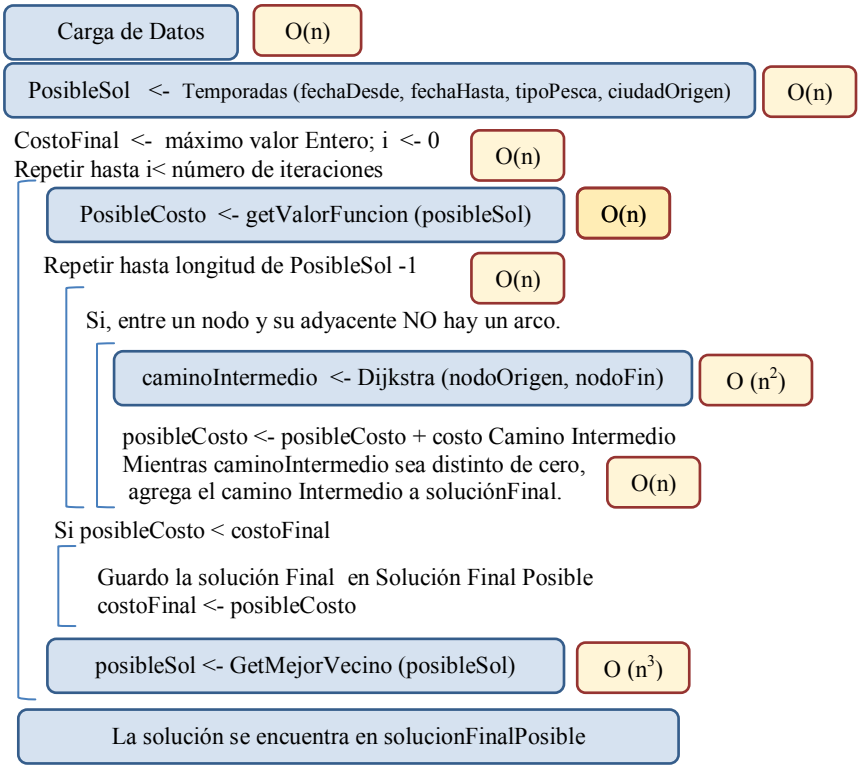


**Figura 3.** Diagrama de flujo y ordenes de ejecución del esquema principal de llamadas, *main*.

## 8 Conclusión

Con este algoritmo resultante se ha comprobado que las heurísticas pueden encontrar, en un tiempo razonable, soluciones aproximadas a la óptima, para la aplicación del TSP a partir de un mapa, los accesos de pesca, una fecha y un tipo de pesca. El método de *búsqueda tabú* se mostró efectivo para este problema

Se modeló el problema en el ámbito de los grafos. Donde los accesos de pesca son los nodos, y las rutas que unen a cada uno de ellos las aristas.

En primer lugar se desarrolló un algoritmo que genere como resultado un arreglo conteniendo aquellos nodos que cumplan con la condición que el usuario ingrese, tipo de pesca, fechas y ciudad origen. Este arreglo es una solución inicial del problema.

En segundo lugar, dada la magnitud y el alcance del problema y dado que su complejidad es *NP-Difícil*, se construyó un algoritmo *metaheurístico de búsqueda tabú* con mejoras iterativas.

Este algoritmo es una mejora sobre la búsqueda local básica, que intentó superar los problemas de la búsqueda local al no quedar atrapados en un mínimo local. Esto permitió contemplar más resultados, intercambiarlos y compararlos. Es por ello que acepta movimientos que no mejoran.

Se desarrolló entonces una heurística que demostró dar muy buenos resultados en muy corto tiempo. Pero el inconveniente inmediato presentado se relacionó con la inexistencia de un camino o arista que conecte dos nodos consecutivos del camino resultante. Esto hizo que el camino sea impracticable en la vida real. Se resolvió este inconveniente mediante la utilización de un algoritmo de *Dijkstra*, el cual devuelve el camino más corto entre dos nodos distintos.

Los resultados finales se ven claramente afectados por la solución inicial ingresada. Si la solución inicial es mala, su resultado final no será el óptimo, debido a que no se llega a realizar todos los intercambios necesarios para llegar a una solución cercana a la óptima. En lo que respecta a los requerimientos, mediante las heurísticas desarrolladas, se ha logrado cumplir con el más importante, la realización de un circuito de pesca óptimo en base a las restricciones planteadas en un comienzo. Los resultados que se obtuvieron son simples, claros y se asemejan al óptimo. Incluyendo ciudades que solo se usarán para conectar con otras que sí serán accesos de pesca, y no solo de paso. Concluyendo, el algoritmo propuesto se encuadra dentro de los Heurísticos Iterativos, y su complejidad es de $O(n^4)$, polinomial.

## Referencias


1. Christofides, N. (1976), "*Worst-case analysis of a new heuristic for the travelling*
2. E.Pesch, F.Glover, Local Search and metaheuristics, in: G. Gutin, A. Punnen (Eds). The Traveling Salesman Problem and Its Variations, Kluwer Academic Publishers, Dordrecht, 2002, pp 309-368.
3. El método de Búsqueda Tabú: http://tesis.uson.mx/digital/tesis/docs/18920/Capitulo2.pdf –Consultado el 10/03/1017
4. Glover, F.; Melián, B. (2003) "Busqueda Tabú", *Revista Iberoamericanda de Inteligencia Artificial* **19:** pp 29-48
5. Johnson, D.S. and McGeoch, L.A.. "*The traveling salesman problem: A case study in local optimization*", Local search in combinatorial optimization, 1997.
6. María Lorena Stockdale .Tesis de Licenciatura - *El problema del viajante: un algoritmo heurístico y una aplicación.*
7. Miguel Sánchez García- *Las Matemáticas del Siglo XX. Optimización Combinatoria.*
8. Rego, César; Gamboa, Dorabela; Glover, Fred; Osterman, Colin. *Traveling salesman problem heuristics: Leading methods, implementations and latest advances* - European Journal of Operational Research. Volume 211, Issue 3, 16 June 2011, Pp 427–441