CAPÍTULO 5: HEURÍSTICAS CONVENCIONALES PARA TSP

5.1. INTRODUCCIÓN

Las propuestas para resolver el problema del viajante de comercio son muchas. Algunas de ellas están basadas en la programación dinámica o en los métodos de *branch and bound*, los cuales proveen la solución óptima global (después de 4 años con más de 7000 ciudades). Otras opciones más rápidas son los *métodos aproximativos*; pero estas no aseguran el hallazgo de la solución óptima [88]. Algunas de ellas pueden ser: los algoritmos basados en operadores de cambio, el algoritmo de *Lin-Kerninghan*, los algoritmos basados en principios voraces (greedy), simulated anneling, *tabu search*, redes neuronales, los algoritmos evolutivos; pero se han aplicado a un número de ciudades menor a 1000 [139]. En las siguientes secciones se describen algunos de los métodos antes mencionados para resolver el problema del viajante. Los algoritmos evolutivos basados en los genéticos, para TSP son descriptos en el próximo capítulo. Los AEs presentan una excelente performance para resolver problemas de optimización, en especial el de scheduling y el de TSP.

5.2. MÉTODOS HEURÍSTICOS DE BÚSQUEDA PARA RESOLVER EL TSP

5.2.1. BRANCH AND BOUND

Branch and Bound es un método de búsqueda general [132]. Usado para optimizar una función $f(x)$, donde x se encuentra restringido a alguna región factible (definida, por ejemplo, por restricciones matemáticas específicas). Para aplicar branch and bound, es necesario desarrollar etapas similares consecutivas, mediante las cuales se llega a detectar la solución buscada en un proceso ordenado de exploración. Cada Etapa esta constituida por dos fases sucesivas que se cumplen en el orden establecido a continuación [102]:

1. Primera Fase: *Branch* (ramificación o separación).

   Dado un problema, se denomina S_1 al conjunto de soluciones s_k. La primera fase consiste en establecer un criterio de partición de ese conjunto

en base a una propiedad dada. En tal forma se obtienen por lo menos dos subconjuntos S_{11} y S_{12} tales que:

$$S_{11} \cup S_{12} = S_1$$
$$S_{11} \cap S_{12} = \varnothing$$

Esto significa que entre ambos subconjuntos totalizan las soluciones del problema original y que los mismos no tienen elementos en común. Si s_k.es una posible solución del problema, resulta que $s_k \in S_1$ y además se cumple que $s_k \in S_{11}$ ó $s_k \in S_{12}$.

El número de soluciones s_k contenidas en S_{11} y S_{12} puede ser distinto para diferentes criterios de partición. Uno de los subconjuntos obtenidos puede ser vacío o contener una o varias soluciones.

En síntesis, la primer fase consiste en efectuar una partición de un conjunto de acuerdo a una propiedad establecida denominada criterio de partición.

2. Segunda Fase: *Bound* (acotación o evaluación y selección).

   Dada, a partir de la formulación del problema, la función objetivo $f(s)$ a optimizar se supone que se desea obtener el mínimo, entonces es posible encontrar un valor de c_{11} cómo cota inferior de la función para todos los $s_{k1} \in S_{11}$; es decir que:

$$c_{11} < f(s_{k1}).$$

   Esto significa que aún cuando no se calcule el $f(s_{k1})$ para todos los $s_{k1} \in S_{11}$ está identificado un valor que los acota inferiormente a todos ellos. Análogamente resulta para todos los $s_{k2} \in S_{12}$, que:

$$c_{12} < f(s_{k2,}).$$

   En consecuencia, si se comparan c_{11} y c_{12}, resulta más favorable para el caso de mínimo, el subconjunto que posea menor cota. Por ejemplo, si

$c_{12} < c_{11}$, es S_{12} el conjunto seleccionado para cumplir con el nuevo proceso de separación.

En resumen, la segunda fase, al establecer la cota inferior de los valores de $f(s)$ para todas las soluciones de un subconjunto cualquiera, permite también realizar la selección de aquel subconjunto que, no habiendo sido particionado, posea la cota más favorable.

Cabe señalar que cuando el subconjunto está compuesto por un solo elemento la cota elegida debe coincidir con el valor que toma la función para dicha solución.

Cumplidas las fases 1 y 2 se completa una etapa. Si se supone que S_{12} es el subconjunto seleccionado y que el mismo contiene más de un elemento, entonces S_{12} pasa a ser la base de una segunda etapa denominándose S_2 e iniciándose nuevamente las fases 1 y 2 respectivamente. El proceso finaliza cuando el subconjunto, que se seleccione por tener la cota mínima entre todos los subconjuntos no particionados posea un único elemento. El mismo define la solución óptima y su cota, coincidente con el valor que toma la función para dicha solución, representa el mínimo de la función en cuestión.

5.2.2. SIMULATED ANNEALING

Simulated annealing es una opción más compleja para la optimización combinatorial, sugerido inicialmente por Metropolis [104]. Esta basado en observaciones del proceso de enfriado de metales. El término annealing se relaciona con la manera en que los metales en estado líquido son enfriados lentamente para asegurar un estado de energía mínima. Se puede decir que el algoritmo de simulated annealing enfría la solución lentamente hasta alcanzar el objetivo más bajo posible [88]. Esto es, en cada una de las evaluaciones de una solución perteneciente al vecindario, siempre toma un movimiento que mejora la solución.

Sin embargo, permite hacer movimientos que incrementen el costo de la solución basado en una función de probabilidad. Esto supera una de las desventajas asociadas a la búsqueda local. En otras palabras, la facultad de realizar movimientos hacia arriba permite escapar de un mínimo local; explorando

así en forma extensiva el espacio de búsqueda. En los pasos descendientes no existe un mecanismo real para salir de un mínimo local. Algunas técnicas para evitar quedar atrapados en estos puntos, incluyen el incremento del tamaño del vecindario y el uso del muestreo aleatorio pero esto no siempre es completamente satisfactorio.

La función de probabilidad depende del cambio en el costo de un movimiento candidato y de la temperatura actual del sistema. La temperatura es análoga a la temperatura en el proceso de enfriado físico. Es decir, la temperatura se reduce a través del proceso de enfriado. A altas temperaturas cualquier movimiento es permitido, lo que posibilita una extensa exploración del espacio de soluciones; mientras que con las bajas se aceptan pocos movimientos hacia arriba. La idea detrás de la reducción de la temperatura, es que más tarde sobre el proceso de enfriado simplemente se explote el vecindario de una solución óptima local. La temperatura se reduce a través de una planificación adecuada de enfriamiento.

Van Laarhoven y Aarts [142] y más recientemente Reeves [115] concluyen que la performance del algoritmo es fuertemente dependiente de la planificación de enfriamiento empleada.

Las comparaciones entre simulated annealing y los métodos tradicionales para problemas de optimización combinatorial han mostrado resultados de diversa calidad. En una de las investigaciones más cabales Johnson [84, 85] muestra que los algoritmos de simulated annealing obtienen resultados superiores a los tradicionales para el problema de coloreo de grafos, mejores sólo en algunos casos para el problema de particionamiento de grafos.

Simulated annealing se presenta como una atractiva técnica de optimización, la cual es aplicable a una amplia variedad de problemas y relativamente simple de implementar.

La figura 5.2 presenta el algoritmo correspondiente al procedimiento de simulated annealing.

```
procedure Simulated Annealing
begin
      n ← 0;
      inicializar temperatura t;
      seleccionar aleatoriamente un string actual T;
      evaluar T;
      repeat
            repeat
              Seleccionar un nuevo string T' en el vecindario de T al mutar un
              único bit de T;
              if f (T) < f (T')
                    then T ← T';
                    else if  random[0,1) < exp{(  f (T') - f (T))/t}
                          then T ← T';
       until (condición de terminación);
        t ← g(t,n);
       n ← n +1;
     until (criterio de detención);
end
```

Figura 5.1. Procedimiento Simulated Anneling

Donde, la condición de terminación chequea si el equilibrio térmico es alcanzado, en algunas implementaciones este bucle se ejecuta un número *m* de veces. La temperatura *t* es reducida gradualmente a través de la función $g(t,n)$, y en algunos casos *t* es reducido cada cierto número de iteraciones *n*. El algoritmo finaliza con un valor pequeño de *t*, el criterio de detención controla si el sistema está congelado de ser así por ejemplo no aceptará más cambios.

Simulated annealing se ha aplicado con éxito en el TSP [19, 84, 87, 89]. Para TSP simulated annealing es significativamente más lento que Lin-Kernighan, pero este tiene la ventaja que se puede ejecutar por largo tiempo y lentamente mejorar la calidad de los resultados [90]. Obteniendo eventualmente resultados comparables o mejores a los brindados por Lin y Kernighan [87].

5.2.3. TABU SEARCH

El concepto básico de Tabu Search es descripto por Glover en 1986 [65] como una meta-heurística impuesta sobre otra heurística. El objetivo es evitar ciclos al prohibir o penalizar movimientos de la solución, a puntos en el espacio de solución previamente visitados (“tabu”). El método Tabu Search ha sido parcialmente motivado por la observación del comportamiento humano con un

elemento aleatorio que conduce a un comportamiento inconsistente en circunstancias similares. Como Glover señala, la tendencia resultante para desviarse desde de un curso programado, debe ser visto como una fuente de error pero también puede ser probada como fuente de beneficios. Tabu Search procede de acuerdo a la suposición de que no existe una restricción para aceptar una nueva (pobre) solución, a menos que la restricción evite un camino ya investigado. Esto asegura que nuevas regiones del espacio de solución del problema serán investigadas con el objetivo de evitar mínimos locales y encontrar la solución deseada.

Este procedimiento guía a una heurística de escalamiento descendiente con el objetivo de continuar la exploración evitando un retroceso a un óptimo local del cual ya se ha salido. En cada iteración se aplica un movimiento admisible a la solución actual, transformándola de ésta manera en una solución vecina con menor costo. Son permitidas las soluciones que incrementan la función de costo, mientras que el movimiento inverso está prohibido para algunas iteraciones con el objetivo de evitar la ocurrencia de ciclos.

Durante el proceso de búsqueda los movimientos son almacenados en una *lista Tabu*, representando la memoria de los pasos previos del algoritmo. Se cuenta con un proceso para determinar cuando las restricciones Tabu pueden ser sobreescritas.

En muchos casos, las diferencias entre las distintas implementaciones del método Tabu están relacionadas con el tamaño, variabilidad y adaptabilidad de la memoria Tabu a un dominio particular.

La figura 5.3 presenta el algoritmo del procedimiento Tabu Search.

```
procedure Tabu-Searh
begin
      x ← solución inicial factible;
      inicializar nmax con número máximo de iteraciones;
      mejor solución ← x ;
      inicializar el número de iteraciones n = 0;
      inicializar la lista Tabu;
     repeat
       elegir un movimiento no tabu factible Δx (n+1);
       x(n+1) ← x(n) + Δx (n+1);
       if (fobj(x(n+1)) > fobj(mejor solución))
            then   mejor solución ← x(n+1);
       actualizar lista Tabu;
       n ← n+1;
    until (n=nmax or cualquier movimiento posible de la solución actual es Tabu);
end
```

Figura 5.2. Procedimiento Tabu-Search

El dominio de aplicaciones de Tabu Search, tradicionalmente, está conformado por los problemas de optimización combinatorial. La técnica se aplica sencillamente a funciones continuas al elegir una codificación discreta del problema. Muchas de las aplicaciones en la literatura involucran: los problemas de programación entera, scheduling, ruteo, TSP y problemas relacionados.

Dada la flexibilidad inherente al método Tabu Search, no sorprende que una amplia variedad de algoritmos Tabu Search se propusieran para el TSP, por ejemplo, ver [50, 65, 66, 67, 79, 91, 92, 101, 120, 141]. No está claro cuál de estos mecanismos es preferible. Todo ellos requieren un tiempo de ejecución $\Omega(N^3)$ y es poco probable que sean prácticos para instancias de gran tamaño.

5.2.4. REDES NEURONALES

Se define a las estructuras de *redes neuronales* como colecciones de procesadores paralelos conectados entre sí en la forma de grafo dirigido, organizado de tal modo que la estructura de la red sea la adecuada para el problema que se este considerando. Es posible representar esquemáticamente cada elemento de procesamiento de la red como un nodo, indicando las conexiones entre nodos mediante arcos.

Una ventaja muy importante de la aproximación de redes neuronales para resolver problemas, consiste en que no es necesario tener un proceso bien definido

para transformar algorítmicamente una entrada en una salida. Mas bien, lo que se necesita para la mayoría de las redes es una colección de ejemplos representativos de la traducción deseada. La red neuronal se adapta, al entrenarse, para reproducir las salidas deseadas cuando se le presentan las entradas dadas como ejemplo.

Además una red neuronal es robusta, en el sentido de que responderá con alguna salida; incluso en el caso de que se presente una entrada que no haya visto nunca, así como entradas que contengan ruido.

El proceso de entrenamiento es una codificación de la información acerca del problema que hay que resolver, y que la red invierte la mayor parte de su existencia productiva siendo ejercitada una vez que concluye el entrenamiento; entonces se descubre un método para permitir que los sistemas automatizados evolucionen sin ser reprogramados explícitamente.

Cuando se describen las bases matemáticas de los modelos de redes, suele resultar útil pensar que la red es un sistema dinámico; esto es, un sistema que evoluciona a lo largo del tiempo.

También resulta útil ver la colección de valores de los pesos comunes a un sistema dinámico (el aprendizaje es un resultado de la modificación de la fuerza de las uniones sinópticas entre neuronas). El *proceso de aprendizaje* consiste en hallar los pesos que codifican el conocimiento que se desea que el sistema aprenda. Para la mayor parte de los sistemas reales, no es fácil determinar una solución en forma cerrada para este sistema de ecuaciones.

La primera aplicación de una red neuronal para el TSP ha sido la dada por Hopfield and Tank en 1985 [83]. La cual está basada en una formulación de la programación entera del TSP [86].

El algoritmo de Hopfield y Tank regularmente encuentra tours óptimos para instancias de 10 ciudades, aunque a menudo no converge en soluciones factibles cuando el número de ciudades $N=30$, en general no brinda resultados promisorios. Además esta opción es muy sensitiva a los pesos de conexión.

Otra forma de representación de una red neuronal para TSP es la geométrica, donde las neuronas pueden ser vistas como un conjunto de $M \geq N$ puntos en el plano, inicialmente ubicados como los vértices de un polígono regular, de M lados, en el medio de la instancia. El objetivo es mover iterativamente estos

vértices hacia las ciudades, esto significa deformar el polígono. Este proceso continúa hasta que el polígono luzca como un tour, y cada ciudad está representada por un vértice, los vértices que no representen a ninguna ciudad están situados en una línea recta entre otros dos, que sí representan ciudades (figura 5.4). Existen dos variantes en este tema, la red elástica de Durbin y Willshaw [33] y *self-organizing map* derivado de las ideas de Kohonen [93].

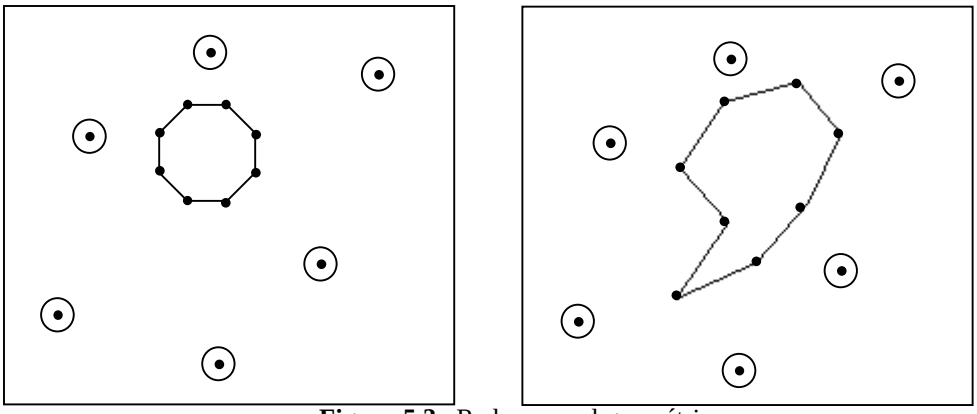

Figura 5.3. Red neuronal geométrica

La primer variante se ha comparado, por sus creadores, con un algoritmo de simulated annealing de calidad desconocida. Siendo los resultados obtenidos para instancias de 50 ciudades un 3% peor que los de ese algoritmo de Simulated Annealing, mientras que para más de 100 ciudades se han observado mejores resultados.

Self-organizing map es una variante de la red elástica, que se ha inspirado en las redes neuronales competitivas de Kohonen [93]. En general los experimentos que se han realizado sobre esta variante concluyen que puede converger más rápidamente que la variante anterior, lo que tiende a producir tours levemente peores [4, 17].

5.2.5. ALGORITMO LIN-KERNIGHAN

Esta es una variante de la búsqueda local en profundidad [103]. Comienza con una noción de la estructura del vecindario de un conjunto de todas las soluciones factibles (tours). Se define el vecindario de un tour, *T* obteniendo los tours

correspondientes al intercambiar a lo sumo k arcos de T. Se inicia con un tour aleatorio T_1 y se construye una secuencia de tours $T_1, T_2,... T_n$. En otras palabras, cada tour es obtenido [99] desde uno previo al realizar *k-cambios*, por ejemplo, al eliminar k enlaces y al reconectar las partes finales desprendidas para lograr un tour. Los *k-cambios* son necesarios para decrementar la longitud del tour. Cuando el proceso se detiene en un tour para el cual no hay posibles mejoras después de *k-cambios*, el tour es *k*-opt. Lin ha introducido y estudiado el caso de $k = 2$ y $k = 3$ y ha mostrado que se pueden obtener tours bastante buenos, rápidamente. Para encontrar un tour óptimo globalmente sugiere repetir la búsqueda desde varios puntos de inicio aleatorios. Posteriormente Lin y Kernighan [100] han realizado una mejora a este algoritmo, la cual consiste en una búsqueda local en ancho (3-Opt) seguida de una búsqueda local en profundidad.