

Algorítmica II. Algoritmos Metaheurísticos

Joaquín Roiz Pagador¹

Universidad Pablo de Olavide, Carretera Utrera s/n, Sevilla, España,
quiniroiz@gmail.com

Abstract. El siguiente documento pertenece a una serie de documentos que pretende servir a modo de resumen para el temario de Algorítmica II para el Grado en Ingeniería Informática en Sistemas de Información.

Keywords: algoritmos, metaheurística, búsqueda, tabú

Búsqueda Tabú.

El algoritmo de búsqueda tabú fue propuesto por Glover en 1986. En 1990, este algoritmo llegó a ser muy popular solventando problemas de optimización de forma aproximada. Hoy en día es uno de los algoritmos metaheurísticos más extendidos.

Comparando con el enfriamiento simulado, podemos observar que esta técnica es menos aleatoria, TS acepta peores movimientos y modifica entornos. También utiliza la memoria adaptativa y estrategias especiales de resolución de problemas.

Principales ventajas de TS:

1. Restringe el entorno de búsqueda.
 - Evitamos recorridos Cíclicos.
 - Generamos vecinos modificados.
2. Introducimos mecanismos de reinicialización.
 - Intensificamos en zonas exploradas.
 - Diversificamos en zonas poco visitadas.

Se utilizarán dos tipos de memoria:

1. Memorias a corto plazo o lista tabú \leftarrow Decisión de vecinos. Guarda información que permite guiar la búsqueda de forma inmediata, desde el comienzo del procedimiento (entornos restringidos/tabú)
2. Memorias a largo plazo \leftarrow Búsqueda de nuevo máximo. Guarda información que permite guiar la búsqueda a *posteriori*, después de una primera etapa en la que se han realizado una o varias ejecuciones del algoritmo aplicando la memoria a corto plazo.

La lista será dinámica, es decir, habrá cambios.

El principio de TS podría resumirse como: "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." (F.Glover)

Memoria + Aprendizaje = Búsqueda inteligente

TS explícitamente usa el historial de búsqueda. Aplica una búsqueda local y usa una memoria a corto plazo para escapar del mínimo local y evitar ciclos. La memoria a corto plazo implementa una lista tabú que almacenará un listado de vecinos visitados con soluciones y prohíbe utilizarlas

nuevamente.

En los siguientes puntos describiremos las memorias a corto y a largo plazo, pero a groso modo podemos decir que, por cada iteración en la lista tabú, a modo de cola de prioridades, un elemento se insertará, y el más antiguo se eliminará, similar a un sistema FIFO.

El algoritmo parará cuando una condición de parada es encontrada, por ejemplo, cuando el número de soluciones prohibidas en la lista tabú es completo.

Una lista tabú de tamaño pequeño hará que se concentre la búsqueda en pequeñas áreas del espacio de búsqueda. Por el contrario, una lista tabú de mayor tamaño fuerza el proceso de búsquedas en regiones más extensas, debido a que esto prohíbe visitar un alto número de soluciones.

Memoria a Corto Plazo

- Modificación de las estructuras de entorno.
 - La TS restringe la búsqueda local sustituyendo $E(S_{act})$ por otro entorno $E^*(S_{act})$. En cada iteración, se acepta siempre el mejor vecino de dicho entorno, tanto si es peor como si es mejor que S_{act} .
 - La memoria de corto plazo (**Lista tabú**) permite a la TS determinar $E^*(S_{act})$ y así organizar la manera en la que se explora el espacio.
- Las soluciones admitidas en $E^*(S_{act})$ dependen de la estructura de la lista tabú:
 - **Lista de soluciones tabú.** Soluciones ya visitadas que se marcan como tabú para no volver a ellas, eliminándolas del vecindario de S_{act} .
 - **Lista de movimientos tabú.** Se eliminan del entorno todos los vecinos resultantes de aplicar sobre S_{act} un movimiento realizado anteriormente.
 - **Lista de valores de atributos tabú.** Se eliminan del entorno todos aquellos vecinos con un par (atributo, valor) determinado que ya presentara alguna solución explorada anteriormente (NOTA: en TSP no nos vendría bien, mejor dependencias como día, clima, tráfico...).
- Tendencia tabú. Tiempo que una solución permanece en la lista tabú.
- Niveles de aspiración: Introduce flexibilidad.
 - Se genera una solución mejor que cualquiera anterior.
 - Se alcanza valor aceptable.
 - Se alcanza el mejor valor conocido.
- Estrategias para listas de candidatos.
 - Restringen el número de vecinos examinados en una iteración dada, para los casos en los que $E^*(S_{act})$ es grande o la evaluación costosa.
 - Se busca el mejor movimiento disponible que pueda ser determinado con esfuerzo equilibrado.
 - Se genera una parte del espacio y se toma el mejor vecino.

Una selección adecuada puede reducir el Tiempo Computacional.

Memoria a Largo Plazo

En algunas aplicaciones, las componentes de la memoria TS de corto plazo son suficientes para producir soluciones de muy alta calidad. Pero TS se vuelve mucho más potente incluyendo memoria de largo plazo. La memoria de largo plazo se puede usar de dos modos:

- Intensificación y diversificación. Memoria de frecuencias y visitas a una solución.
- Reinicializar la búsqueda, mantener registro de mejores soluciones.. Se pueden usar para diversificar.
- Flowchart \rightarrow Serie de iteraciones a corto plazo, donde extraemos la mejor de ellas.
- Nueva solución a partir de lo almacenado, evitaremos repetir.

Una estructura muy empleada es la memoria de frecuencias, que registra el número de veces que cada valor de un atributo ha pertenecido a soluciones visitadas en la búsqueda.

Estrategias de Intensificación

- Se basan en una reinicialización de la búsqueda que efectúa un regreso a regiones atractivas del espacio para buscar en ellas más extensamente.
- Se mantiene un registro de las mejores soluciones visitadas, insertando una nueva solución cada vez que se convierte en la mejor.
- Se puede introducir una medida de diversificación para asegurar que las soluciones registradas difieran una de otra en un grado deseado.

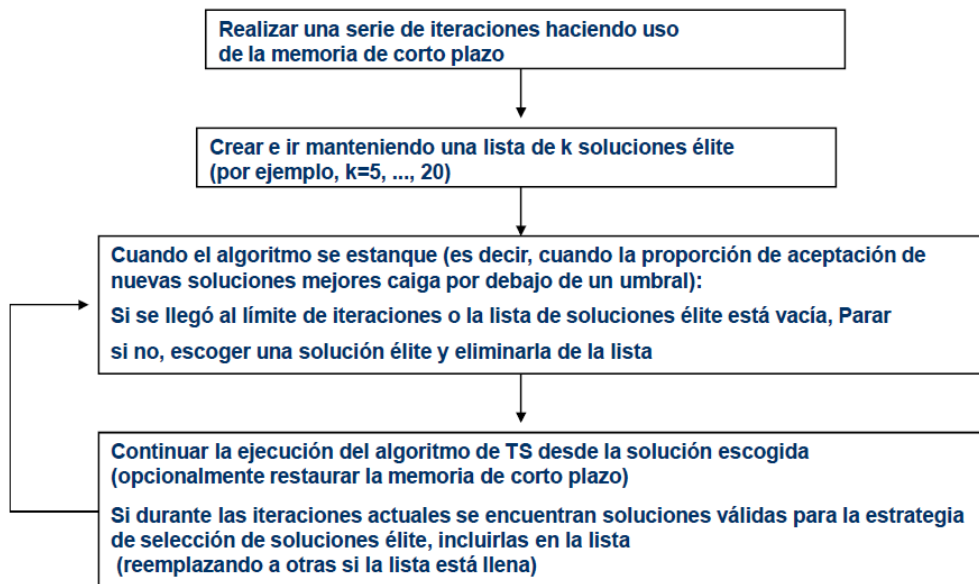


Fig. 1. Flowchart Intensificación

Estrategias de Diversificación

- Conducen la búsqueda hacia nuevas regiones del espacio de búsqueda no exploradas aún.
- La búsqueda se reinicializa cuando se estanca, partiendo de una solución no visitada.
- Esta solución se genera a partir de la memoria de frecuencias, dando mayor probabilidad de aparición a los valores menos habituales.

Estructuras de memoria largo plazo

- Si las variables son binarias, se puede usar un vector de dimensión n , para almacenar el número de veces que cada variable tomó el valor 0 (ó 1).
- Si son enteras, se utiliza una matriz bidimensional, como contador de las veces que la variable i toma el valor k : $m[i,k]$.
- Si son permutaciones de orden, se puede utilizar una matriz bidimensional, como contador de las veces que el valor i ha ido seguido de j : $M[i,j]$

Uso de las memorias de frecuencias

1. Generar directamente la nueva solución inicial a partir de la información almacenada en la memoria de frecuencias M , dando mayor probabilidad de aparición a los valores menos habituales.

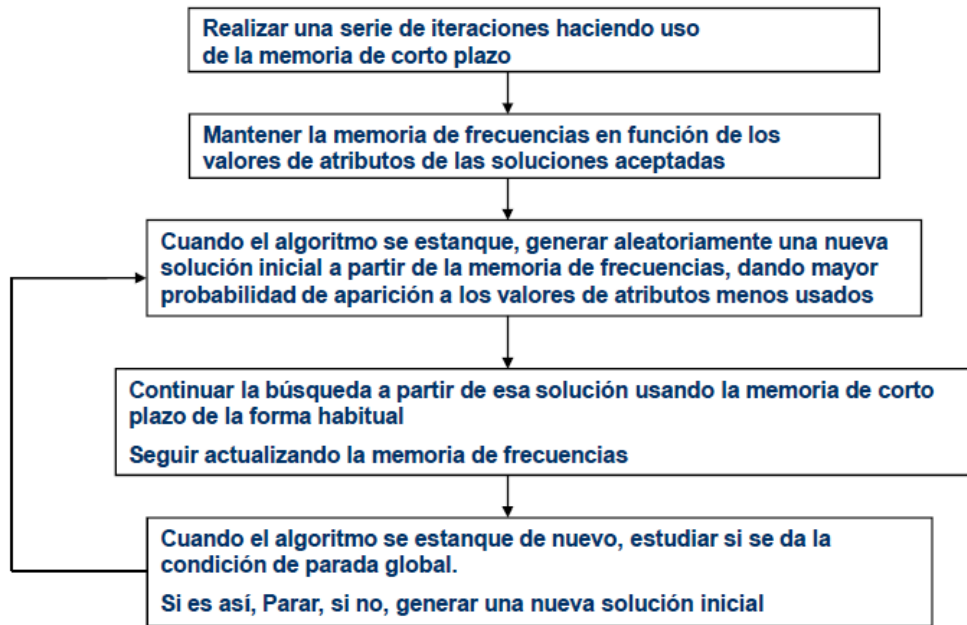


Fig. 2. Flowchart Diversificación

2. Usar la información almacenada en M para modificar temporalmente el caso del problema, potenciando los valores heurísticos de los atributos menos usados en la búsqueda. Aplicar un algoritmo greedy sobre ese caso modificado para generar la solución inicial. Restaurar el caso original del problema antes de continuar con la búsqueda.

```

sBest ← s0
tabuList ← []
while notstoppingCondition() do
  candidateList ← []
  bestCandidate ← null
  for sCandidate in sNeighborhood do
    if (not tabuList.contains(sCandidate)) and (fitness(sCandidate) >
      fitness(bestCandidate)) then
      bestCandidate ← sCandidate
    end
  end
  if fitness(bestCandidate) > fitness(sBest) then
    sBest ← bestCandidate
  end
  tabuList.push(bestCandidate)
  if tabuList.size > maxTabuSize then
    tabuList.removeFirst()
  end
end
output: Best Solution found.

```

Algorithm 1: Algoritmo de Búsqueda Tabú con memoria a corto plazo.

Si quisiéramos añadir una condición de aspiración, en la condición perteneciente a la iteración de los vecinos deberíamos añadir "or sCandidate.isAspiration()".

Aplicación de temario: [http://leeds-faculty.colorado.edu/glover/fred%20pubs/329%20-%20Introduccion%20a%20la%20Busqueda%20Tabu%20TS_Spanish%20w%20Belen\(11-9-06\).pdf](http://leeds-faculty.colorado.edu/glover/fred%20pubs/329%20-%20Introduccion%20a%20la%20Busqueda%20Tabu%20TS_Spanish%20w%20Belen(11-9-06).pdf)

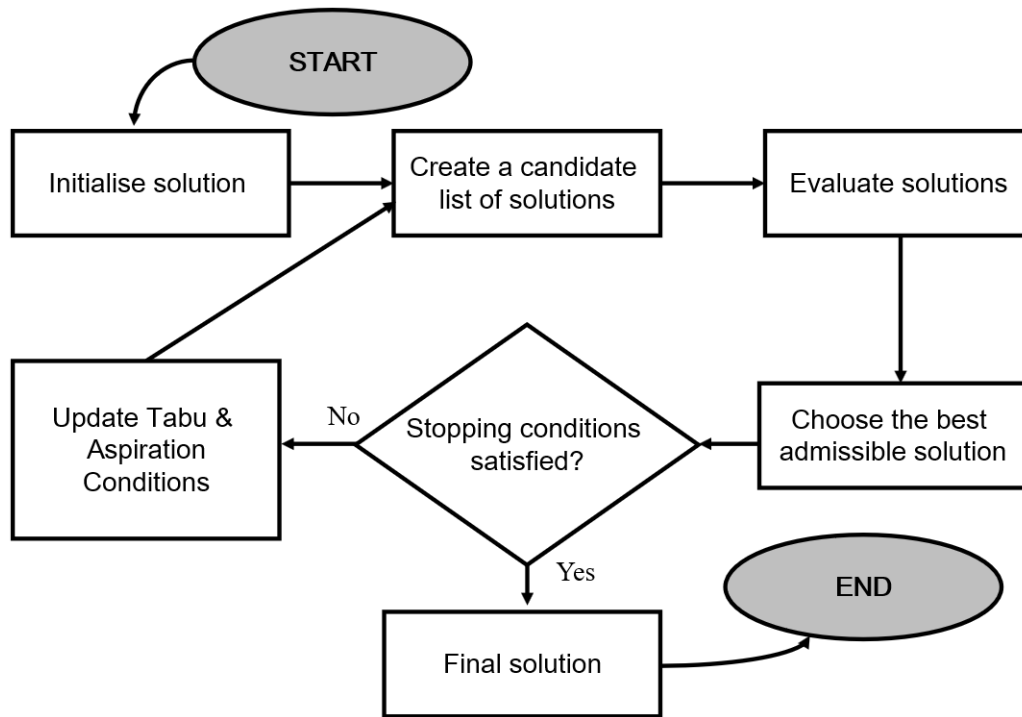


Fig. 3. Diagrama de flujo Búsqueda Tabú

References

1. Material asignatura, *Metaheurísticas*, UGR. <http://sci2s.ugr.es/graduateCourses/Metaheurísticas>
2. E.-G. Talbi, *Metaheuristics. From design to implementation*, Wiley, 2009 Material.
3. C. Blum, A Roli, *Metaheuristics in Combinatorial Optimization: overview and conceptual comparison*. *ACM Computing Surveys*, 35(3), 2003, 268-308.
4. B. Melián, F. Glover. *Introducción a la búsqueda tabú*, 2006. [http://leeds-faculty.colorado.edu/glover/fred%20pubs/329%20-%20Introduccion%20a%20la%20Busqueda%20Tabu%20TS_Spanish%20w%20Belen\(11-9-06\).pdf](http://leeds-faculty.colorado.edu/glover/fred%20pubs/329%20-%20Introduccion%20a%20la%20Busqueda%20Tabu%20TS_Spanish%20w%20Belen(11-9-06).pdf)