Backtracking y Branch and Bound

Problema del viajante de comercio

A. Herrera, A. Moya, I. Sevillano, J.L. Suarez

27 de mayo de 2015

Introducción

- En esta presentación se resuelve de forma óptima el problema del viajante de comercio mediante ***Backtracking*** y ***Branch and Bound***.
- El código, los resultados de las ejecuciones, las gráficas y los pdf asociados se puede encontrar en GitHub.

Explicación del problema

Enunciado

Construir un programa que utilice la técnica de ramificación y acotación para resolver el problema del viajante de comercio en las condiciones descritas anteriormente, empleando una determinada función de acotación.

Opcionalmente, construir también un programa que utilice vuelta atrás, pero utilizando también la función de acotación descrita anteriormente, y realizar un estudio experimental comparativo con el algoritmo de ramificación y poda.

Explicación del problema

A. Herrera, A. Moya, I. Sevillano, J.L. Suarez

Información sobre el problema

- El problema del viajante de comercio es NP-Hard.
- ¡Una instancia con n ciudades tiene $(n-1)!$ posibles ciclos hamiltonianos!.
- Un algoritmo de fuerza bruta difícilmente resuelve el problema para n mayor que 11.

Algoritmos propuestos

A. Herrera, A. Moya, I. Sevillano, J.L. Suarez

- Buscamos hallar el óptimo con un mejor rendimiento que el algoritmo de fuerza bruta. Recurrimos a la **poda**.

Algoritmos

1. *Backtracking*
2. *Branch and Bound*

Backtracking

A. Herrera, A. Moya, I. Sevillano, J.L. Suarez

Conceptos clave

- Modelizar el espacio de soluciones como las hojas de un **árbol**.
- **Recorrer el árbol** en profundidad.
- Aplicar un criterio de **poda**.

A. Herrera, A. Moya, I. Sevillano, J.L. Suarez

Árbol de soluciones

- Un **nodo** del árbol consiste en una solución parcial al problema, esto es, un camino de ciudades no repetidas.
- Un **hijo de un nodo** consiste en añadir una ciudad no visitada al final del camino.
- Cada nodo tiene tantos hijos como ciudades no estén en su camino.
- Un nodo será **hoja** cuando su camino contenga todas las ciudades del problema. En tal caso, consideramos el ciclo obtenido de cerrar su camino como solución.
- El **nodo raíz** tiene un camino con una única ciudad (aleatoria).

Backtracking

Recorrido en profundidad

- Comenzando por la raíz, se visitan sus hijos de forma recursiva tal y como un recorrido en **post-orden**.
- Puede ser implementado recursivamente o mediante una **pila** (LIFO).

A. Herrera, A. Moya, I. Sevillano, J.L. Suarez

Poda

Proposición 1. Si el coste del ciclo formado por cerrar el camino de un nodo es mayor que el coste de la mejor solución obtenida se puede podar la rama.

Demostración. La distancia del camino del nodo se mantendrá en cualquier solución futura de la rama. La distancia añadida al cerrar el ciclo es menor o igual que la distancia que se recorrerá al visitar las ciudades restantes y cerrarlo (desigualdad triangular).

A. Herrera, A. Moya, I. Sevillano, J.L. Suarez

Solución inicial

- Generamos una **solución inicial** voraz mediante el algoritmo del vecino más cercano para podar desde un principio.

Backtracking. Pseudocódigo

A. Herrera, A. Moya, I. Sevillano, J.L. Suarez

```
def TSPBacktracking(candidatos, sol, mejor_sol)
  if candidatos.empty and sol < mejor_sol
    mejor_sol = sol
  end
  for i in 0..candidatos.size
    sol.pushCiudad(candidatos[i])
    candidatos.eraseAt(i)
    if sol < mejor_sol
      TSPBacktracking(candidatos,sol, mejor_sol)
    end
    candidatos[i] = sol.popCiudad()
  end
```

Backtracking

A. Herrera, A. Moya, I. Sevillano, J.L. Suarez

Pros

- Sencillo de comprender e implementar.
- Muchos mejores resultados que una fuerza bruta clásica.

Contras

- Poda manifiestamente mejorable.
- Se desarrollan ramas innecesarias por culpa del recorrido en profundidad.

Branch and Bound

A. Herrera, A. Moya, I. Sevillano, J.L. Suarez

Conceptos clave

- Modelizar el espacio de soluciones como las hojas de un **árbol**.
- Obtener una **cota inferior** para cada nodo.
- Recorrer el árbol mediante una **cola con prioridad**.
- **Podar** utilizando la cota inferior.

Branch and Bound

Árbol de soluciones (análogo a Backtracking)

- Un **nodo** del árbol consiste en una solución parcial al problema, esto es, un camino de ciudades no repetidas.
- Un **hijo de un nodo** consiste en añadir una ciudad no visitada al final del camino.
- Cada nodo tiene tantos hijos como ciudades no estén en su camino.
- Un nodo será **hoja** cuando su caminocontenga todas las ciudades del problema. En tal caso, consideramos el ciclo definido por el mismo como solución.
- El **nodo raíz** tiene un camino con una única ciudad (aleatoria).

Branch and Bound

Cota inferior

- Para la **primera ciudad** del camino, añadir la menor distancia entre esta y las ciudades no visitadas.
- Para la **última ciudad visitada**, añadir la menor distancia entre esta y las no visitadas.
- Para todas las **ciudades no visitadas** menos una, puesto que en cualquier solución de la rama se va a visitar, añadir la menor distancia entre esta y otra ciudad no visitada.

Branch and Bound

Recorrido mediante cola con prioridad

- Mantenemos una **cola con prioridad para nodos** que se inicializa con la raíz del árbol.
- La cola con prioridad devuelve el nodo con **menor estimación** sin desarrollar.
- Mientras la cola no esté vacía y el frente tenga mejor estimación que la mejor solución obtenida:
  1. Se **desarrollan los hijos** del frente de la cola y se elimina este.
  2. Se calcula la **estimación** de cada hijo.
  3. Si la estimación de un hijo es menor que el coste de la mejor solución obtenida **se añade a la cola**.

Branch and Bound

Poda

- **Criterio de poda:** cota inferior.
- El recorrido mediante cola con prioridad mantiene **implícita** la poda:
  1. Si un hijo de un nodo tiene cota inferior mayor que nuestro mejor coste se poda (no se añade a la cola).
  2. Si un nodo termina siendo inservible según nuestro criterio de poda nunca será frente de la pila.

Branch and Bound

Solución inicial

- Generamos una **solución inicial** voraz mediante el algoritmo del vecino más cercano para podar desde un principio.

Branch and Bound. Pseudocódigo

A. Herrera, A. Moya, I. Sevillano, J.L. Suarez

```
def TSPBranchAndBround()
  #Cola con prioridad con la raíz
  # Inicializamos mejor solución con greedy.
  queue = priority_queue.new
  queue.push(Nodo.new(ciudades.primeraCiudad))
    mejor_solucion = TSPVecinoMasCercano()

  # Mientras haya nodos útiles en la cola
  while not queue.empty() and
    queue.top().estimacion < mejor_solucion

    n = queue.pop # Extraemos primer nodo
    hijos = n.getHijos() # Obtenemos sus hijos
```

Branch and Bound. Pseudocódigo


```
    # Añadimos los hijos útiles
    for hijo in hijos
      hijo.estimar()
      if hijo.estimacion < mejor_solucion
        queue.push(hijo)
      end
    end

    # Si era una hoja se compara con la mejor
    if n.ciudadesVisitadas == totalCiudades
      and n.solucion < mejor_solucion
        mejor_solucion = n.solucion
    end
  end
end
```

Branch and Bound

A. Herrera, A. Moya, I. Sevillano, J.L. Suarez

Pros

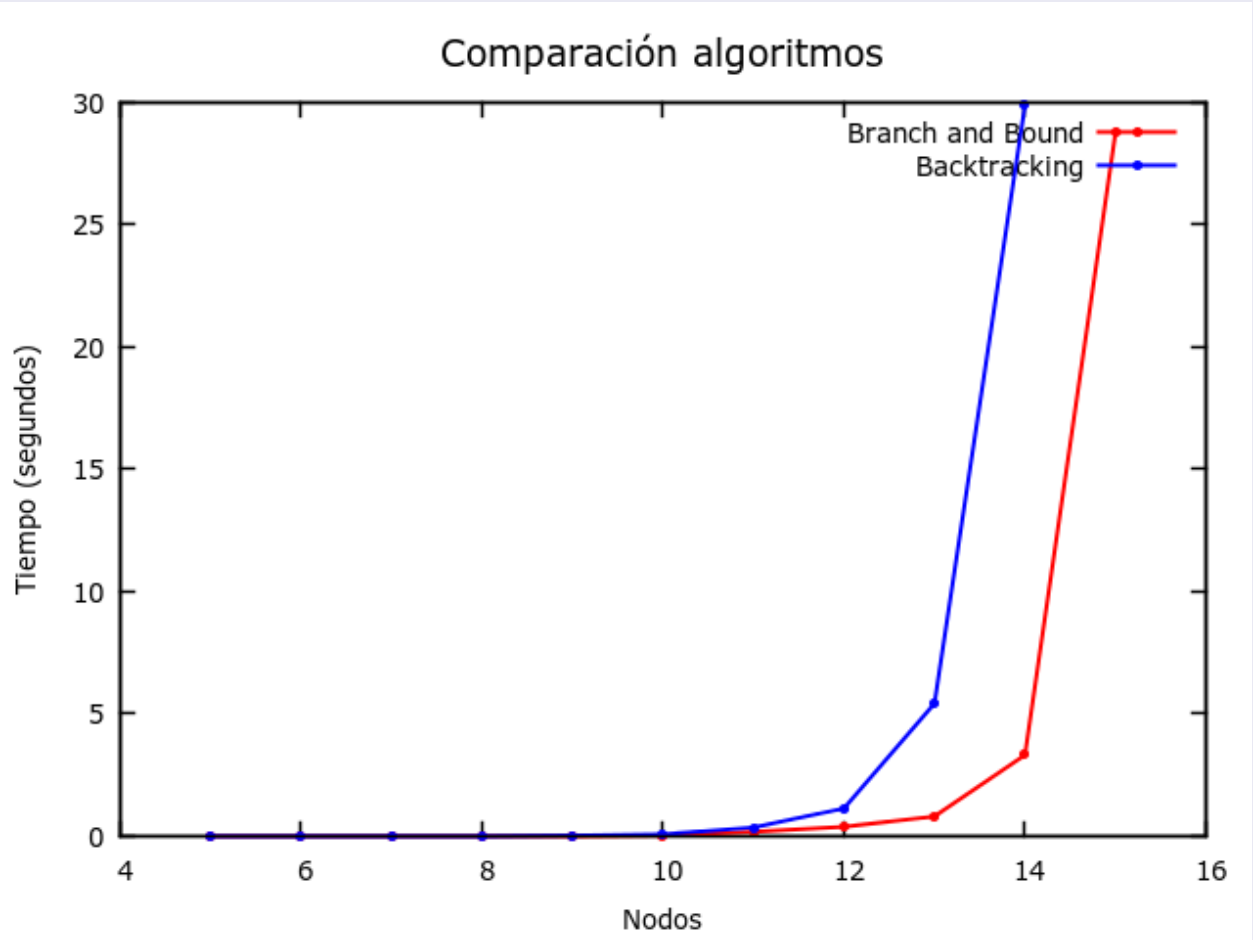
- Solo desarrolla los nodos estrictamente necesarios.
- Gran potencial mejorando cotas inferiores e implementación.
- Mejores resultados que backtracking.

Contras

- Difícil implementación.
- Tiempo de ejecución usado en mantener la cola y hallar estimaciones.

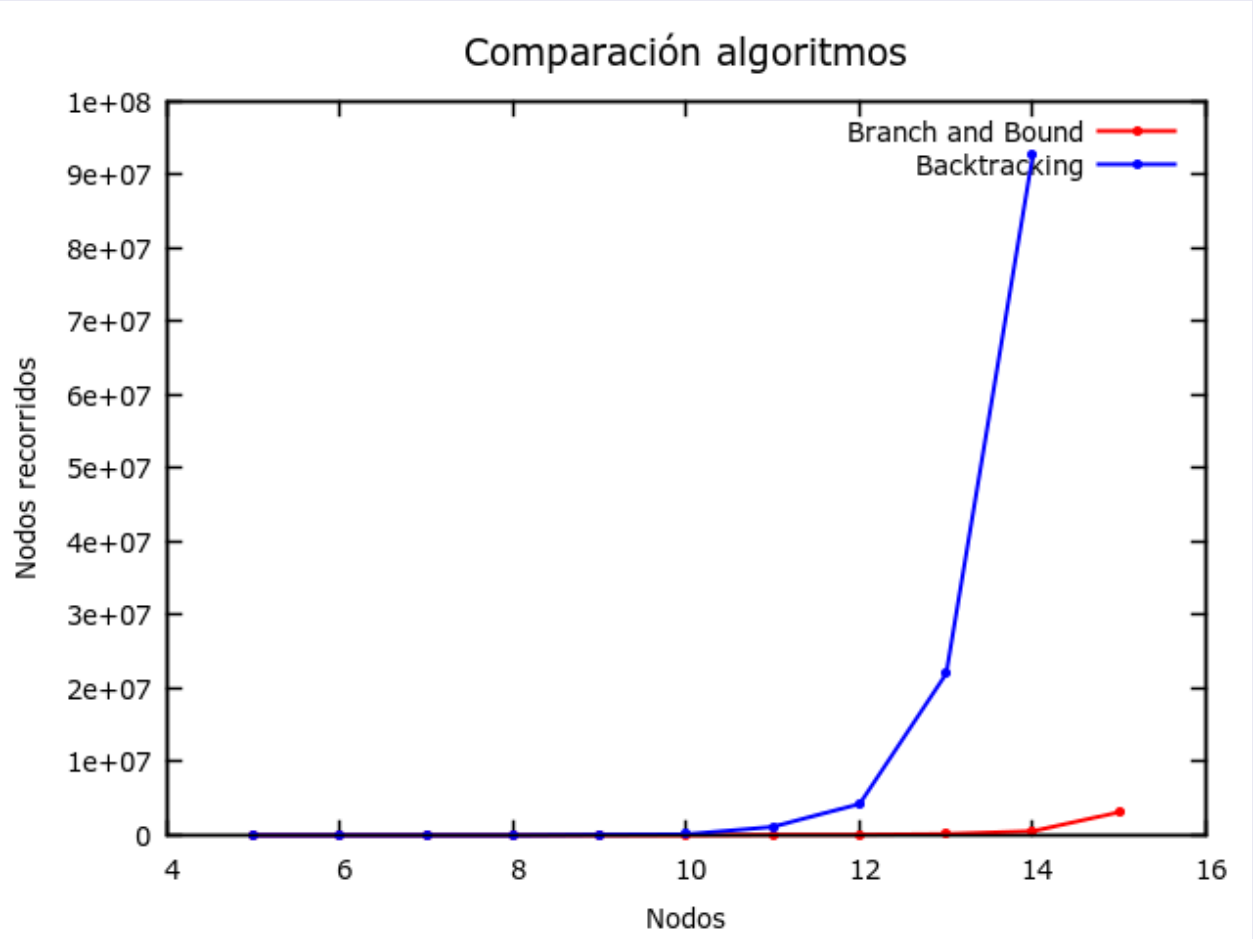
Análisis empírico

Tiempos obtenidos



Análisis empírico

Nodos visitados



Fin de la presentación

¡Gracias por su atención!

¡Arriba la Informática Teórica!