Grafos Eulerianos y Hamiltonianos

Algoritmos y Estructuras de Datos III

Grafos eulerianos

Definiciones:

- Un circuito C en un grafo (o multigrafo) G es un **circuito euleriano** si C pasa por todos las aristas de G una y sólo una vez.

- Un **grafo euleriano** es un grafo que tiene un circuito euleriano (o multigrafo).

Grafos eulerianos

Definiciones:

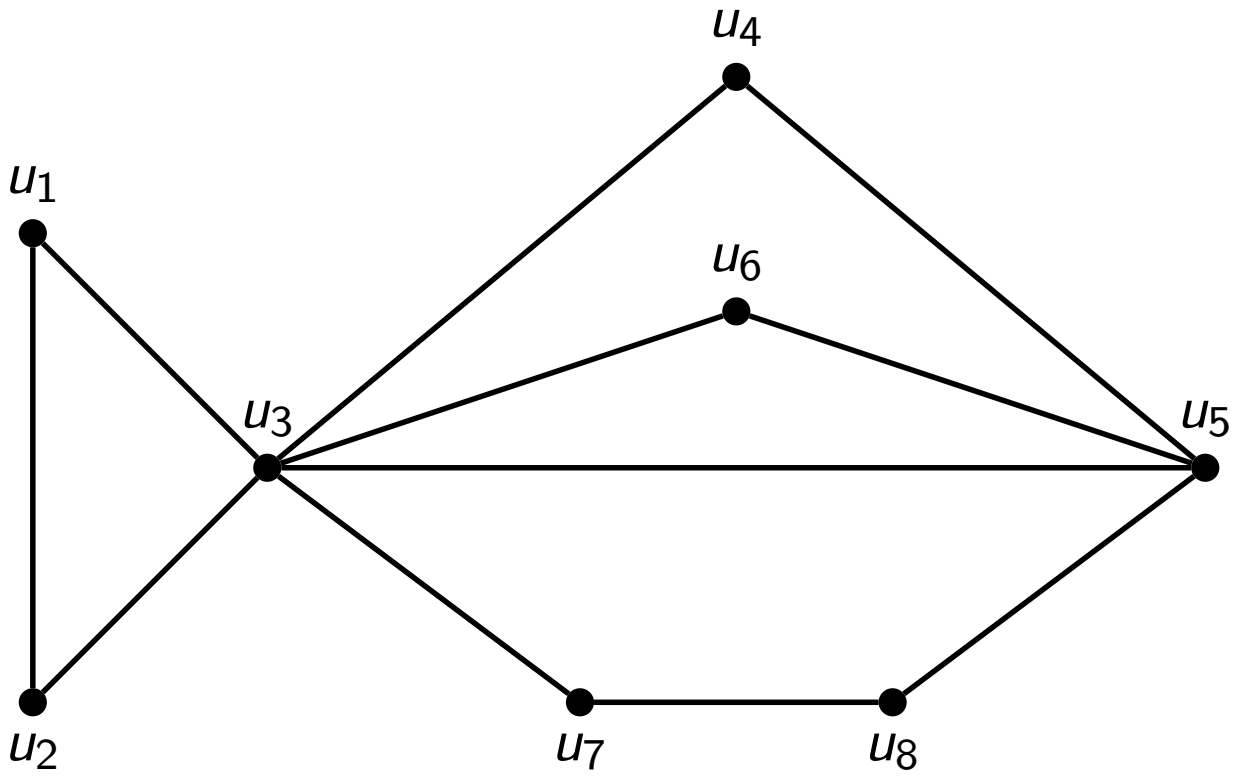
- Un circuito C en un grafo (o multigrafo) G es un **circuito euleriano** si C pasa por todos las aristas de G una y sólo una vez.

- Un **grafo euleriano** es un grafo que tiene un circuito euleriano (o multigrafo).

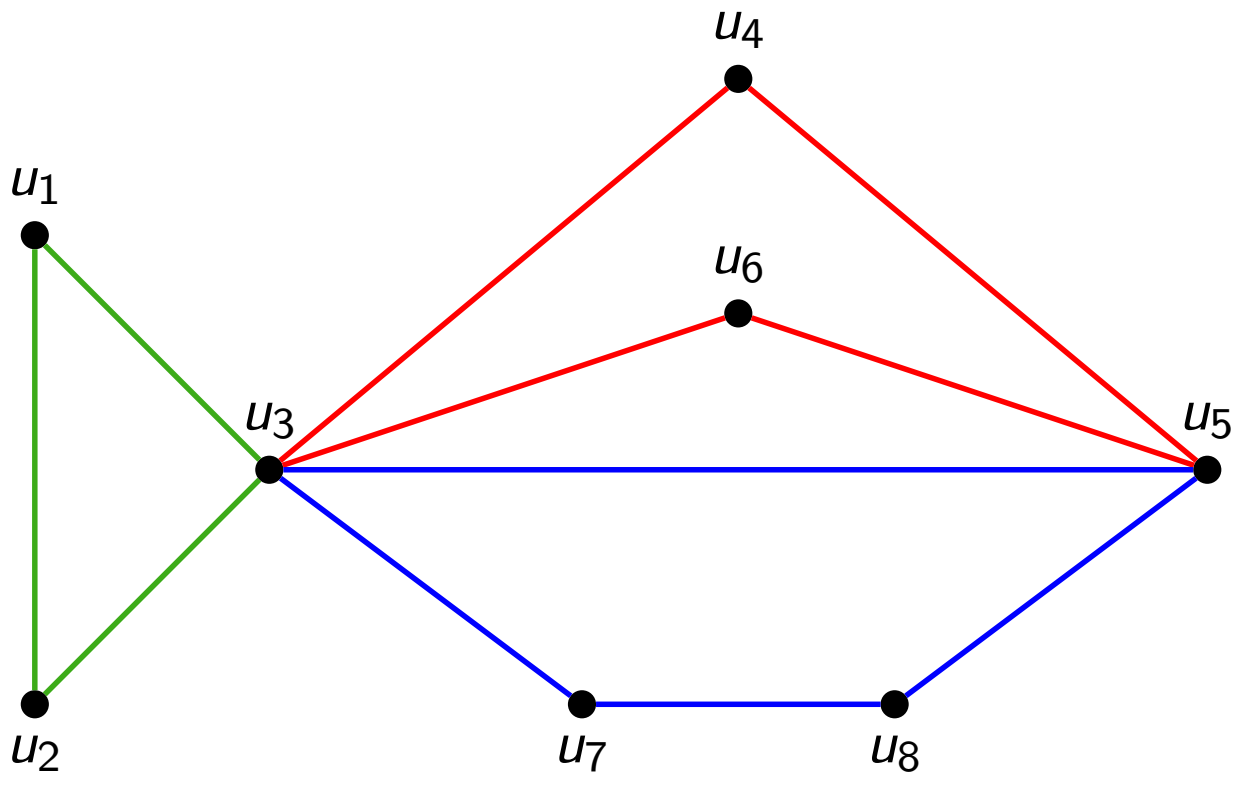
Teorema (Euler 1736): Un grafo (o multigrafo) conexo es euleriano si y sólo si todos sus nodos tienen grado par.

A partir de la demostración del teorema de Euler se puede escribir un algoritmo para construir un circuito euleriano para un grafo que tiene todos sus nodos de grado par.

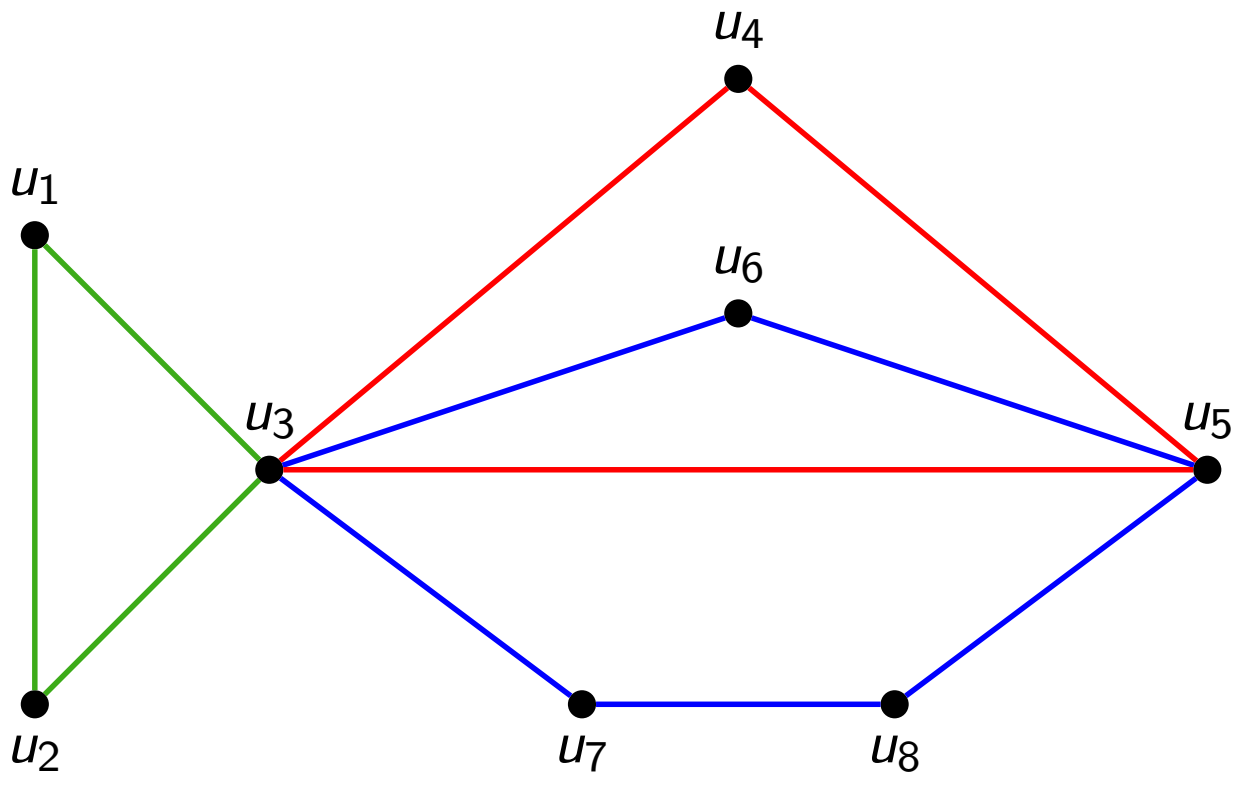
Grafos eulerianos



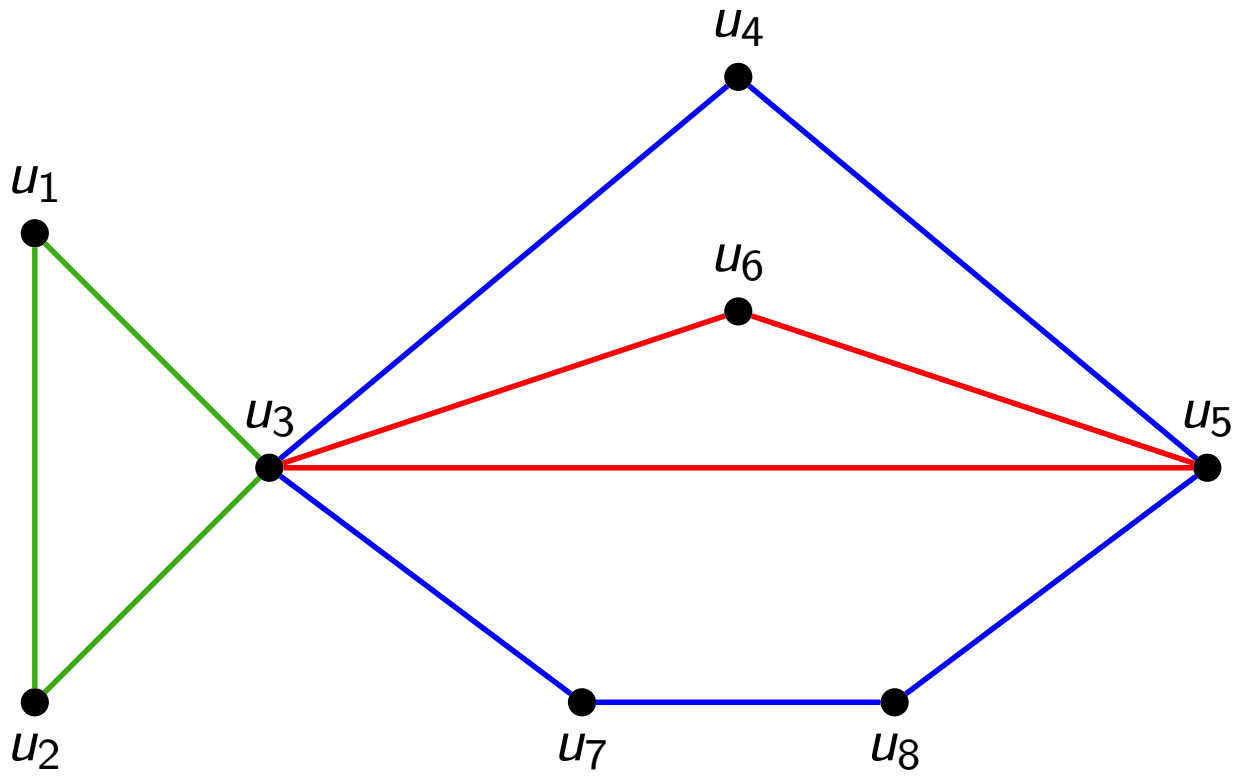
Grafos eulerianos



Grafos eulerianos



Grafos eulerianos



Grafos eulerianos

Entrada: $G = (V, X)$ conexo con todos los nodos de grado par.

comenzar por cualquier nodo v y construir un ciclo Z
mientras exista $e \in X \setminus Z$ **hacer**
    elegir w tal que existe $(w, u) \in Z$ y $(w, z) \in X \setminus Z$
    desde w construir un ciclo D con $D \cap Z = \emptyset$
    $Z :=$ unir Z y D por medio de w
fin mientras
retornar Z

¿Cuál es la complejidad de este algoritmo?

Grafos eulerianos

Definiciones:

- Un **camino euleriano** en un grafo (o multigrafo) G es un camino que pasa por cada arista de G una y sólo una vez.
- Un grafo orientado o digrafo, se dice **euleriano** si tiene un circuito orientado que pasa por cada arco de G una y sólo una vez.

Grafos eulerianos

Definiciones:

- Un **camino euleriano** en un grafo (o multigrafo) G es un camino que pasa por cada arista de G una y sólo una vez.
- Un grafo orientado o digrafo, se dice **euleriano** si tiene un circuito orientado que pasa por cada arco de G una y sólo una vez.

Teorema: Un grafo (o multigrafo) conexo tiene un camino euleriano si y sólo si tiene exactamente dos nodos de grado impar.

Grafos eulerianos

Definiciones:

- Un **camino euleriano** en un grafo (o multigrafo) G es un camino que pasa por cada arista de G una y sólo una vez.
- Un grafo orientado o digrafo, se dice **euleriano** si tiene un circuito orientado que pasa por cada arco de G una y sólo una vez.

Teorema: Un grafo (o multigrafo) conexo tiene un camino euleriano si y sólo si tiene exactamente dos nodos de grado impar.

Teorema: Un digrafo conexo es euleriano si y sólo si para todo nodo v de G se verfica que $d_{in}(v) = d_{out}(v)$.

Problema del cartero chino (Guan, 1962)

Definición: Dado un grafo $G = (V, X)$ con longitudes asignadas a sus aristas, $l : X \to \mathbb{R}^{\geq 0}$, el **problema del cartero chino** consiste en encontrar un circuito que pase por cada arista de G **al menos** una vez de longitud mínima.

Problema del cartero chino (Guan, 1962)

Definición: Dado un grafo $G = (V, X)$ con longitudes asignadas a sus aristas, $l : X \to \mathbb{R}^{\geq 0}$, el **problema del cartero chino** consiste en encontrar un circuito que pase por cada arista de G **al menos** una vez de longitud mínima.

- Si G es euleriano, un circuito euleriano es la solución del problema del cartero chino.
- Hay algoritmos polinomiales para el problema del cartero chino cuando G es orientado o no orientado.
- Pero no se conocen algoritmos polinomiales (el problema no está computacionalmente resuelto) si el grafo es mixto (algunas aristas orientados y otros no).

Grafos hamiltonianos

Definiciones:

- Un circuito en un grafo G es un **circuito hamiltoniano** si pasa por cada nodo de G una y sólo una vez.
- Un grafo se dice **hamiltoniano** si tiene un circuito hamiltoniano.

Grafos hamiltonianos

Definiciones:

- Un circuito en un grafo G es un **circuito hamiltoniano** si pasa por cada nodo de G una y sólo una vez.
- Un grafo se dice **hamiltoniano** si tiene un circuito hamiltoniano.

No se conocen buenas caracterizaciones para grafos hamiltonianos.

Grafos hamiltonianos

Definiciones:

- Un circuito en un grafo G es un **circuito hamiltoniano** si pasa por cada nodo de G una y sólo una vez.
- Un grafo se dice **hamiltoniano** si tiene un circuito hamiltoniano.

No se conocen buenas caracterizaciones para grafos hamiltonianos.

¿Cómo intentar construir un circuito hamiltoniano?

Grafos hamiltonianos

Definiciones:

- Un circuito en un grafo G es un **circuito hamiltoniano** si pasa por cada nodo de G una y sólo una vez.
- Un grafo se dice **hamiltoniano** si tiene un circuito hamiltoniano.

No se conocen buenas caracterizaciones para grafos hamiltonianos.

¿Cómo intentar construir un circuito hamiltoniano?

No se conocen algoritmos polinomiales para decidir si un grafo es hamiltoniano o no.

Grafos hamiltonianos

Teorema (condición necesaria): Sea G un grafo conexo. Si existe $W \subset V, W \neq \emptyset$ tal que $G \setminus W$ tiene c componentes conexas con $c > |W|$ entonces G no es hamiltoniano.

¿Es cierta la recíproca de este teorema?

Grafos hamiltonianos

Teorema (condición necesaria): Sea G un grafo conexo. Si existe $W \subset V, W \neq \emptyset$ tal que $G \setminus W$ tiene c componentes conexas con $c > |W|$ entonces G no es hamiltoniano.

¿Es cierta la recíproca de este teorema?

Teorema (Dirac) (condición suficiente): Sea G un grafo con $n \geq 3$ y tal que para todo $v \in V$ se verifica que $d(v) \geq n/2$ entonces G es hamiltoniano.

¿Es cierta la recíproca de este teorema?

Algoritmos heurísticos

- Un algoritmo heurístico (también llamado una heurística) es un algoritmo que no garantiza una respuesta exacta para el problema en cuestión.

Algoritmos heurísticos

- Un algoritmo heurístico (también llamado una heurística) es un algoritmo que no garantiza una respuesta exacta para el problema en cuestión.

  1. Heurísticas *ad hoc* o "clásicas".
  2. Metaheurísticas.

Algoritmos heurísticos

- Un algoritmo heurístico (también llamado una heurística) es un algoritmo que no garantiza una respuesta exacta para el problema en cuestión.

  1. Heurísticas *ad hoc* o "clásicas".
  2. Metaheurísticas.

- ¿Cuándo es conveniente recurrir a una heurística?

Algoritmos heurísticos

- Un algoritmo heurístico (también llamado una heurística) es un algoritmo que no garantiza una respuesta exacta para el problema en cuestión.
    1. Heurísticas *ad hoc* o "clásicas".
    2. Metaheurísticas.
- ¿Cuándo es conveniente recurrir a una heurística?
    1. Problemas para los cuales no se conocen algoritmos exactos eficientes.
    2. Problemas difíciles de modelar.

Algoritmos heurísticos

- Un algoritmo heurístico (también llamado una heurística) es un algoritmo que no garantiza una respuesta exacta para el problema en cuestión.
  1. Heurísticas *ad hoc* o "clásicas".
  2. Metaheurísticas.
- ¿Cuándo es conveniente recurrir a una heurística?
  1. Problemas para los cuales no se conocen algoritmos exactos eficientes.
  2. Problemas difíciles de modelar.
- ¿Cómo se evalúa una heurística?

Algoritmos heurísticos

- Un algoritmo heurístico (también llamado una heurística) es un algoritmo que no garantiza una respuesta exacta para el problema en cuestión.
  1. Heurísticas *ad hoc* o "clásicas".
  2. Metaheurísticas.
- ¿Cuándo es conveniente recurrir a una heurística?
  1. Problemas para los cuales no se conocen algoritmos exactos eficientes.
  2. Problemas difíciles de modelar.
- ¿Cómo se evalúa una heurística?
  1. Problemas test.
  2. Problemas reales.
  3. Problemas generados al azar.
  4. Cotas inferiores.

Problema del viajante de comercio (TSP)

Definición: Dado un grafo $G = (V, X)$ con longitudes asignadas a las aristas, $l : X \to \mathbb{R}^{\geq 0}$, queremos determinar un circuito hamiltoniano de longitud mínima.

Problema del viajante de comercio (TSP)

Definición: Dado un grafo $G = (V, X)$ con longitudes asignadas a las aristas, $l : X \to \mathbb{R}^{\geq 0}$, queremos determinar un circuito hamiltoniano de longitud mínima.

- No se conocen algoritmos polinomiales para resolver el problema del viajante de comercio.
- Tampoco se conocen algoritmos ϵ-aproximados polinomiales para el TSP general (sí se conocen cuando las distancias son euclideanas).
- Es el problema de optimización combinatoria más estudiado.

Heurísticas y algoritmos aproximados para el TSP

Heurística del vecino más cercano

```
elegir un nodo v
orden(v) := 0
S := {v}
i := 0
mientras S ≠ V hacer
    i := i + 1
    elegir la arista (v, w) más barata con w ∉ S
    orden(w) := i
    S := S ∪ {w}
    v := w
fin mientras
retornar orden
```

¿Cuál es la complejidad de este algoritmo?

Heurísticas y algoritmos aproximados para el TSP

Heurísticas de inserción

$C :=$ un circuito de longitud 3
$S := \{$nodos de $C\}$
mientras $S \neq V$ **hacer**
    ELEGIR un nodo $v \notin S$
    $S := S \cup \{v\}$
    INSERTAR v en C
fin mientras
retornar C

Heurísticas y algoritmos aproximados para el TSP

Heurísticas de inserción

$C :=$ un circuito de longitud 3
$S := \{$nodos de $C\}$
mientras $S \neq V$ **hacer**
    ELEGIR un nodo $v \notin S$
    $S := S \cup \{v\}$
    INSERTAR v en C
fin mientras
retornar C

- ¿Cómo ELEGIR?
- ¿Cómo INSERTAR?

Heurísticas y algoritmos aproximados para el TSP

Heurísticas de inserción

Para ELEGIR e INSERTAR el nodo v elegido:

- Sea $c_{v_i v_j}$ el costo de la arista (v_i, v_j).
- Elegimos dos nodos consecutivos en el circuito v_i, v_{i+1} tal que $c_{v_i v} + c_{v v_{i+1}} - c_{v_i v_{i+1}}$ sea mínimo.
- Insertamos v entre v_i y v_{i+1}.

Heurísticas y algoritmos aproximados para el TSP

Heurísticas de inserción

Podemos ELEGIR el nuevo nodo v para agregar al circuito tal que:

- v sea el nodo más próximo a un nodo que ya está en el circuito.

Heurísticas y algoritmos aproximados para el TSP

Heurísticas de inserción

Podemos ELEGIR el nuevo nodo v para agregar al circuito tal que:

- v sea el nodo más próximo a un nodo que ya está en el circuito.
- v sea el nodo más lejano a un nodo que ya está en el circuito.

Heurísticas y algoritmos aproximados para el TSP

Heurísticas de inserción

Podemos ELEGIR el nuevo nodo v para agregar al circuito tal que:

- v sea el nodo más próximo a un nodo que ya está en el circuito.
- v sea el nodo más lejano a un nodo que ya está en el circuito.
- v sea el nodo *más barato*, o sea el que hace crecer menos la longitud del circuito.

Heurísticas y algoritmos aproximados para el TSP

Heurísticas de inserción

Podemos ELEGIR el nuevo nodo v para agregar al circuito tal que:

- v sea el nodo más próximo a un nodo que ya está en el circuito.
- v sea el nodo más lejano a un nodo que ya está en el circuito.
- v sea el nodo *más barato*, o sea el que hace crecer menos la longitud del circuito.
- v se elige al azar.

Heurísticas y algoritmos aproximados para el TSP

Heurísticas de inserción

En el caso de grafos euclideanos (por ejemplo grafos en el plano $\mathbb{R}^2$), se puede implementar un algoritmo de inserción:

- Usando la cápsula convexa de los nodos como circuito inicial.
- Insertando en cada paso un nodo v tal que el ángulo formado por las aristas (w, v) y (v, z), con w y z consecutivos en el circuito ya construido, sea máximo.

Hay muchas variantes sobre estas ideas.

Heurísticas y algoritmos aproximados para el TSP

Heurística del árbol generador

encontrar un árbol generador mínimo T de G

duplicar las aristas de T

armar un circuito euleriano E con los ejes de T y sus *duplicados*

recorrer E usando **DFS** y armar un circuito hamiltoniano de G

¿Cuál es la complejidad de este algoritmo?

Heurísticas y algoritmos aproximados para el TSP

Heurística del árbol generador

Teorema: Sea C^H el circuito generado por la heurística del árbol generador y sea C^* un circuito óptimo. Si las distancias cumplen la desigualdad triangular, entonces

$$\frac{l(C^H)}{l(C^*)} \leq 2.$$

Heurísticas y algoritmos aproximados para el TSP

Heurística del árbol generador

Teorema: Sea C^H el circuito generado por la heurística del árbol generador y sea C^* un circuito óptimo. Si las distancias cumplen la desigualdad triangular, entonces

$$\frac{l(C^H)}{l(C^*)} \leq 2.$$

Una heurística con estas propiedades se denomina un algoritmo aproximado.

Heurísticas y algoritmos aproximados para el TSP

Perfomance de otros algoritmos aproximados en el peor caso

Si las distancias de G son euclideanas se puede probar que valen las siguientes cotas para la perfomance en el peor caso:

Vecino más próximo: $\frac{l(C^H)}{l(C^*)} \leq 1/2(\lceil \log n \rceil + 1)$

Inserción del más próximo: $\frac{l(C^H)}{l(C^*)} \leq 2$

Inserción del más lejano: $\frac{l(C^H)}{l(C^*)} \leq 2 \log n + 0.16$

Inserción del más barato: $\frac{l(C^H)}{l(C^*)} \leq 2$

Heurísticas y algoritmos aproximados para el TSP

Heurísticas de mejoramiento - Algoritmos de búsqueda local

¿Cómo podemos mejorar la solución obtenida por alguna heurística constructiva como las anteriores?

Heurísticas y algoritmos aproximados para el TSP

Heurísticas de mejoramiento - Algoritmos de búsqueda local

¿Cómo podemos mejorar la solución obtenida por alguna heurística constructiva como las anteriores?

Heurística 2-opt de Lin y Kernighan

`obtener una solución inicial` H
mientras `sea posible` **hacer**

`elegir` (u_i, u_{i+1}) `y` $(u_k, u_{k+1}) \in H$ `tal que`
$c_{u_i u_{i+1}} + c_{u_k u_{k+1}} > c_{u_i u_k} + c_{u_{i+1} u_{k+1}}$

$H := H \setminus \{(u_i, u_{i+1}), (u_k, u_{k+1})\} \cup \{(u_i, u_k), (u_{i+1}, u_{k+1})\}$

fin mientras

Heurísticas y algoritmos aproximados para el TSP

Heurísticas de mejoramiento - Algoritmos de búsqueda local

¿Cómo podemos mejorar la solución obtenida por alguna heurística constructiva como las anteriores?

Heurística 2-opt de Lin y Kernighan

obtener una solución inicial H
mientras sea posible **hacer**

elegir (u_i, u_{i+1}) y $(u_k, u_{k+1}) \in H$ tal que
$c_{u_i u_{i+1}} + c_{u_k u_{k+1}} > c_{u_i u_k} + c_{u_{i+1} u_{k+1}}$

$H := H \setminus \{(u_i, u_{i+1}), (u_k, u_{k+1})\} \cup \{(u_i, u_k), (u_{i+1}, u_{k+1})\}$

fin mientras

¿Cuándo para este algoritmo? ¿Se obtiene la solución óptima del TSP de este modo?

Heurísticas y algoritmos aproximados para el TSP

Heurísticas de mejoramiento - Algoritmos de búsqueda local

- En vez de elegir para sacar de H un par de aristas cualquiera que nos lleve a obtener un circuito de menor longitud podemos elegir, entre todos los pares posibles, el par que nos hace obtener el menor circuito (más trabajo computacional).

- Esta idea se extiende en las heurísticas k-opt donde se hacen intercambios de k aristas. Es decir, en vez de sacar dos aristas, sacamos k aristas de H y vemos cual es la mejor forma de reconstruir el circuito. En la práctica se usa sólo 2-opt o 3-opt.

Algoritmos de descenso o búsqueda local

Esquema general

S = conjunto de soluciones
$N(s)$ = soluciones "vecinas" de la solución s
$f(s)$ = valor de la solución s

elegir una solución inicial $s^* \in S$
repetir
    elegir $s \in N(s^*)$ tal que $f(s) < f(s^*)$
    reemplazar s^* por s
hasta que $f(s) > f(s^*)$ para todos los $s \in N(s^*)$

Algoritmos de descenso o búsqueda local

- ¿Cómo determinar las soluciones vecinas de una solución s dada?

Algoritmos de descenso o búsqueda local

- ¿Cómo determinar las soluciones vecinas de una solución s dada?
- ¿Qué se obtiene con este procedimiento? ¿Sirve?

Algoritmos de descenso o búsqueda local

- ¿Cómo determinar las soluciones vecinas de una solución s dada?
- ¿Qué se obtiene con este procedimiento? ¿Sirve?
- Óptimos locales y globales

Algoritmos de descenso o búsqueda local

- ¿Cómo determinar las soluciones vecinas de una solución s dada?
- ¿Qué se obtiene con este procedimiento? ¿Sirve?
- Óptimos locales y globales
- Espacio de búsqueda

Algoritmos de descenso o búsqueda local

Ejemplo

- Tenemos que asignar n tareas a un sola máquina.
- Cada trabajo j tiene un tiempo de procesamiento p_j y una fecha prometida de entrega d_j.
- El objetivo es minimizar

$$T = \sum_{j=1}^{n} \max\{(C_j - d_j), 0\}$$

donde C_j es el momento en que se completa el trabajo j.

Algoritmos de descenso o búsqueda local

Ejemplo

- **Cómo elegir las soluciones iniciales:** Se podría tomar cualquier permutación de las tareas.
- **Determinación de los vecinos de una solución dada:** Podemos tomar las que se obtengan de la solución actual cambiando la osición de un trabajo con otro.

Algoritmos de descenso o búsqueda local

Ejemplo

- **Cómo elegir las soluciones iniciales:** Se podría tomar cualquier permutación de las tareas.
- **Determinación de los vecinos de una solución dada:** Podemos tomar las que se obtengan de la solución actual cambiando la osición de un trabajo con otro.

Por ejemplo, en un problema con 4 trabajos, los vecinos de $s = (1, 2, 3, 4)$ serán:

$$N(s) = \{(1, 3, 2, 4), (3, 2, 1, 4), (1, 2, 4, 3), (1, 4, 3, 2), (2, 1, 3, 4), (4, 2, 3, 1)\}$$

GRASP

Esquema general

T. Feo y M. Resende, *Greedy randomized adaptive search procedures*. Journal of Global Optimization (1995) 1–27.

mientras `no se verifique el criterio de parada` **hacer**

```
solución:=construirGreedyRandomizedSolución
mejorSolución:=búsquedaLocal(solución)
actualizarSolución(solución,mejorSolución)
```

GRASP

Esquema general

- **Algoritmo construirGreedyRandomizedSolución**: En vez de usar un algoritmo goloso que elija el elemento más prometedor, según una función "adaptativa", para agregar a la solución, en cada iteración se elige al azar entre los que cumplen que no pasan de un porcentaje α del valor del mejor elemento. Se puede limitar el tamaño de la lista de estos elementos.

- **Algoritmo búsquedaLocal**: Definición de intercambios.

GRASP

Ejemplo: Cubrimiento de conjuntos

- Dados n conjuntos $P_1, P_2, \ldots, P_n$.
- Sea $I = \cup_i P_i$ y $J = \{1, 2, \ldots, n\}$.
- Un subconjunto J^* de J es un **cubrimiento** si $\cup_{i \in J^*} P_i = I$.
- El problema de **recubrimiento mínimo** (set covering problem) consiste en determinar un cubrimiento de I de cardinal mínimo (con la mínima cantidad de conjuntos P_i).

GRASP

Ejemplo: Cubrimiento de conjuntos

Dados $P_1 = \{1, 2\}$, $P_2 = \{1, 3\}$, $P_3 = \{2\}$, $P_4 = \{3\}$

$$I = \{1, 2, 3\} \qquad J = \{1, 2, 3, 4\}$$

Los cubrimientos mínimos tienen cardinal 2 y son:

$$\{P_1, P_2\} \qquad \{P_1, P_4\} \qquad \{P_2, P_3\}$$

GRASP

Ejemplo: Cubrimiento de conjuntos

Primer paso: `solución := ConstruirGreedyRandomizedSolución`

Un algoritmo goloso podría ser agregar al cubrimiento el conjunto que cubre la mayor cantidad de elementos de *I* **todavía** sin cubrir.

En este caso para la parte del algoritmo `GreedyRandomized` consideramos como conjuntos candidatos a los que cubren al menos un porcentaje α del número cubierto por el conjunto determinado por el algoritmo goloso.

También se puede limitar el tamaño de la lista de candidatos a tener a lo sumo β elementos.

Dentro de esta lista de conjuntos se elige uno al azar.

GRASP

Ejemplo: Cubrimiento de conjuntos

Segundo paso: `mejorSolución := búsquedaLocal(solución)`

Para el algoritmo de descenso podemos definir los vecinos de una solución usando el siguiente procedimiento de intercambios:

Un k, p-intercambio, con $p < k$, consiste en cambiar, si es posible, k-uplas del cubrimiento por p-uplas que no pertenezcan al mismo.