# Material de apoyo Teórica XIV

## Temario

- Heurísticas para el problema del viajante

Soluciones heurísticas para el problema del viajante.

***Solución Exacta:***

***El problema del viajante de comercio pertenece a la clase de problemas* NP-Hard*. (NP = No-determinístico polinomial).***

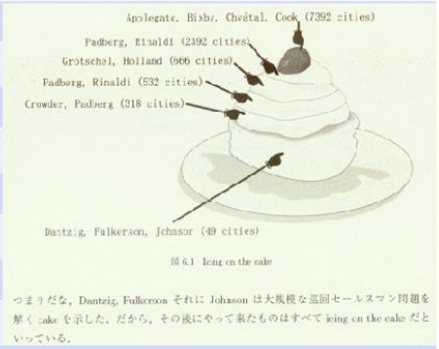
***Todos los problemas de esta clase son difíciles. Cualquier algoritmo que se conoce para resolver el problema lleva un número exponencial de pasos (Complejidad exponencial = $2^n$).***

***Hay n! Soluciones para cada viajante (donde n es la cantidad de ciudades)***

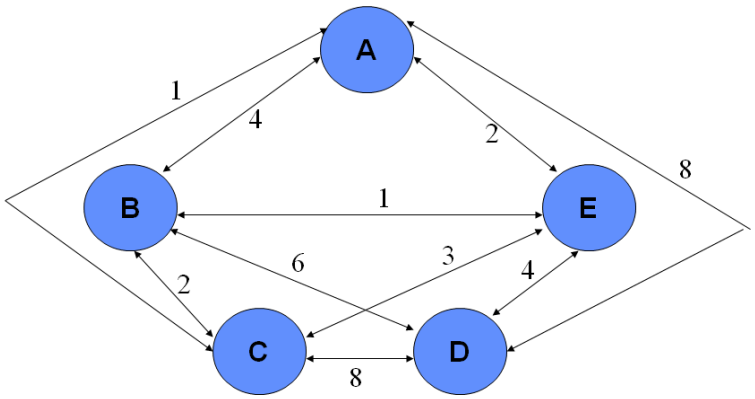
**El problema del viajante.**

***Solución exacta:***

| Año | Tamaño de la instancia |
|---|---|
| 1954 | 49 |
| 1971 | 64 |
| 1975 | 67 |
| 1977 | 120 |
| 1980 | 318 |
| 1987 | 532 / 666 / 2.392 |
| 1994 | 7.397 |
| 1998 | 13.509 |
| 2001 | 15.112 (ciudades de Alemania) |
| 2004 | 24.978 (ciudades de Suecia) |
| 2006 | 85.900 (puntos de soldadura sobre un circuito impreso) |



Veamos un ejemplo de un viajante simétrico que parte desde A:




***<u>Heurística del vecino más próximo</u>:***

- ***Se empieza por un tour parcial trivial que contiene una sola ciudad. (Heurística para elegir el punto de partida).***

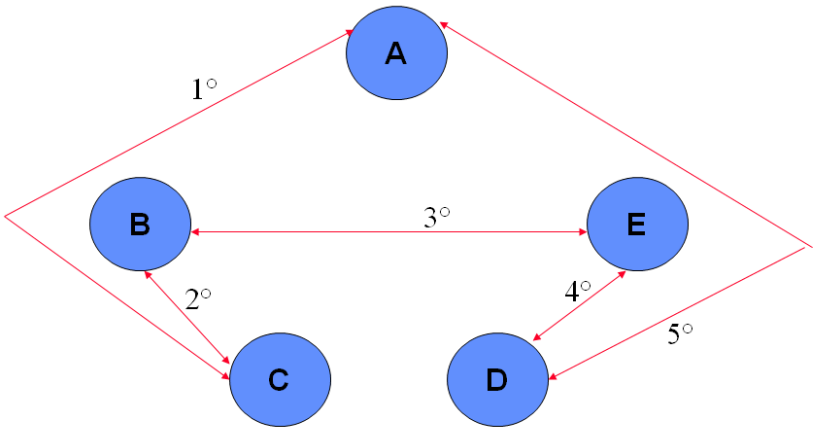
***Mientras queden ciudades sin agregar al tour:***

- ***La próxima ciudad elegida es la más cercana a la última del tour siempre que no esté ya incluida en el tour (si hay más de una que sea la más cercana, elegir por orden alfabético).***

***Fin mientras***

- ***Cuando todas las ciudades están en el tour se conecta la última con el origen.***

Heurística del vecino más próximo




CTO = 16

***<u>Heurística del emparchamiento más cercano</u>:***

- ***Elegir el eje con menor costo *(si hay más de uno, elegir por orden alfab.).***

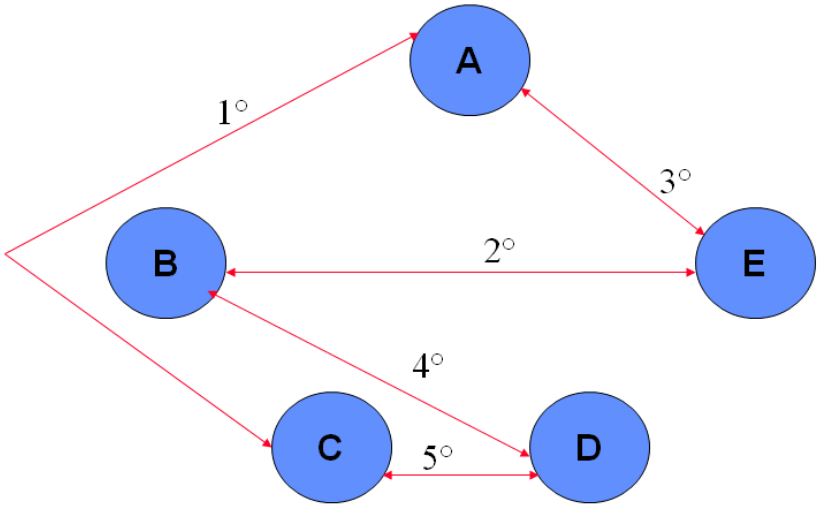
***Mientras queden ciudades sin agregar al tour:***

- ***Ir eligiendo de los ejes restantes el que tenga menor costo (*) y nos permita seguir agregando ciudades (el eje se agrega en cualquiera de los extremos).***

***Fin mientras***

- ***Se agrega el eje que permita unir las ciudades que hayan quedado desconectadas.***

Heurística de emparchamiento más cercano




CTO = 18

***<u>Heurística de Inserción más cercana</u>:***

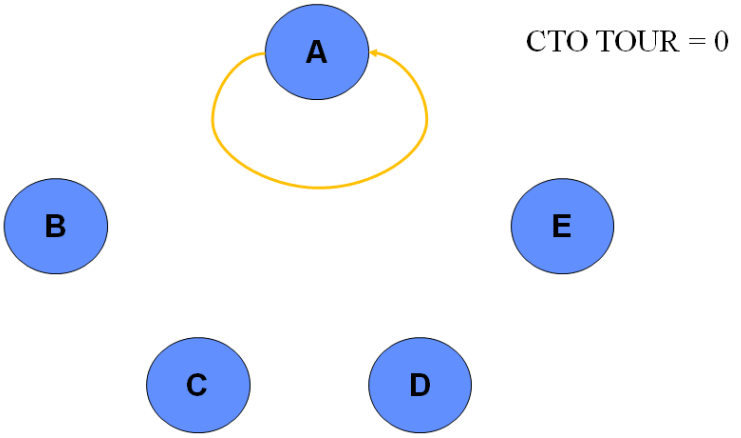
- ***Comenzar con un subtour de una sola ciudad.***

***Mientras queden ciudades sin agregar al tour:***

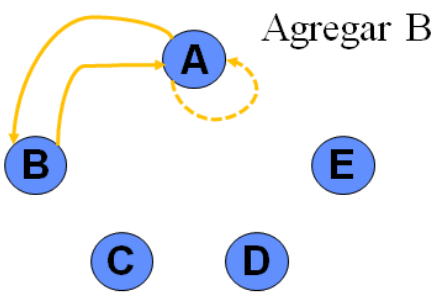
- ***La próxima ciudad a agregar (que aún no haya sido visitada) es la que permita armar un nuevo subtour, con una ciudad más, al menor costo posible. Para cada candidata a agregar se calcula la diferencia de costo entre el tour anterior y el obtenido con el agregado de esta nueva ciudad.***

***Fin mientras***

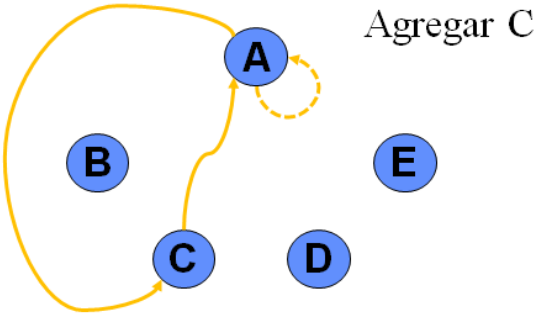
**Heurística de inserción más cercana**



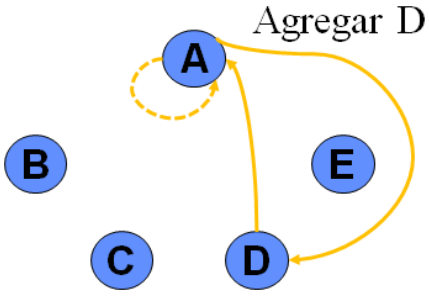

***Vamos a ver cuál de las otras cuatro ciudades conviene agregar al tour***



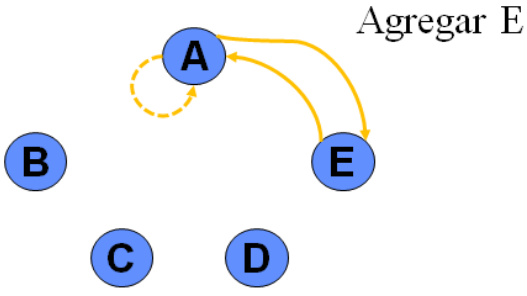

**CTO TOUR = 0 - 0 + (4 + 4) = 8**




**CTO TOUR = 0 - 0 + (1 + 1) = 2**



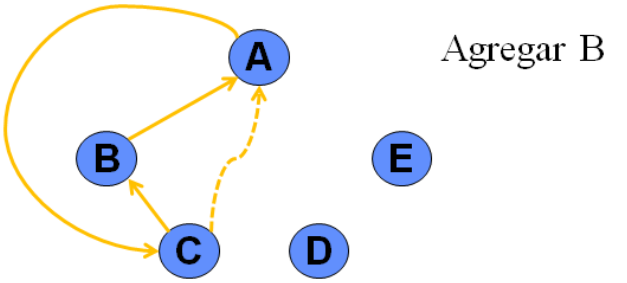

**CTO TOUR = 0 - 0 + (8 + 8) = 16**



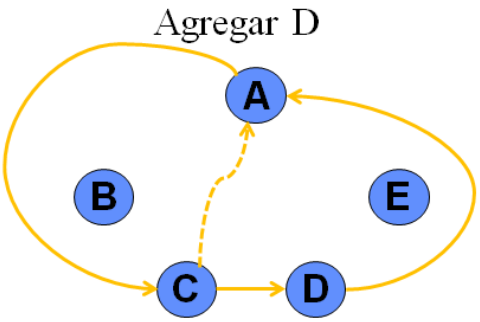

**CTO TOUR = 0 - 0 + (2 + 2) = 4**

**Entonces, agregamos C y el tour nos quedó cerrado A-C-A**

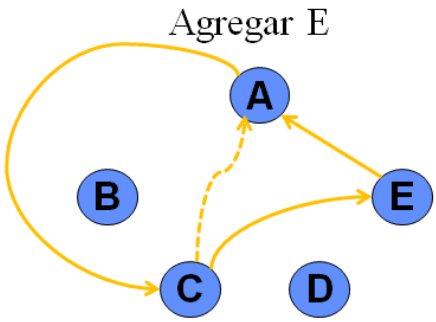
**Ahora vamos a ver si conviene agregar B, E o D al tour y en qué posición**




**CTO TOUR = 2 - 1 + (2 + 4) = 7**




**CTO TOUR = 2 - 1 + (8 + 8) = 17**




**CTO TOUR = 2 - 1 + (3 + 2) = 6**

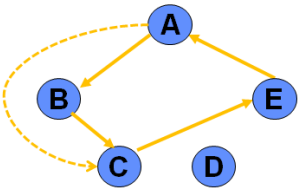
**Entonces, agregamos E y el tour nos queda cerrado A-C-E-A**

**Ahora vamos a ver si conviene agregar B o D al tour y en qué posición**

**Heurística de inserción más cercana**

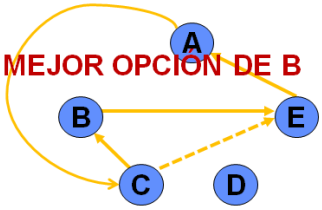
Probemos de agregar B

Entre A y C



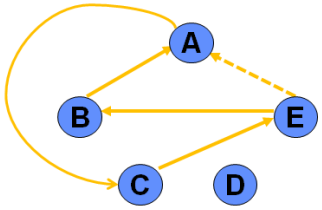

**CTO TOUR = 6 - 1 + (4 + 2) = 11**

Entre C y E




**CTO TOUR = 6 - 3 + (1 + 2) = 6**

Entre E y A

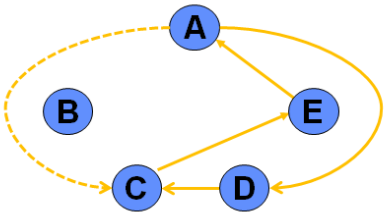



**CTO TOUR = 6 - 2 + (3 + 1) = 8**

**Heurística de inserción más cercana**

Probemos de agregar D

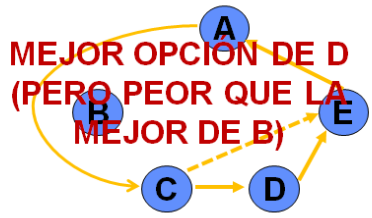
Entre A y C




**CTO TOUR = 6 - 1 + (8 + 8) = 21**

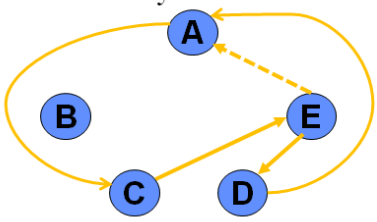
Entre C y E

MEJOR OPCIÓN DE D (PERO PEOR QUE LA MEJOR DE B)




**CTO TOUR = 6 - 3 + (8 + 4) = 15**

Entre E y A

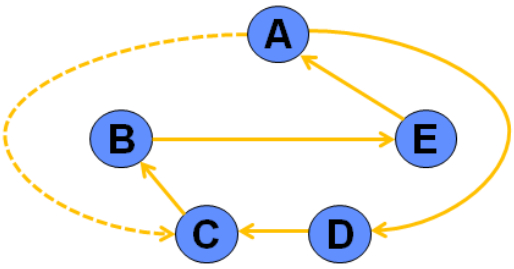



**CTO TOUR = 6 - 2 + (4 + 8) = 16**

**Entonces, agregamos B y el tour nos queda cerrado A-C-B-E-A**

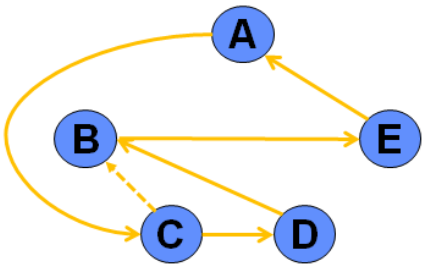
**Ahora ya sabemos que nos queda agregar D al tour pero tenemos que ver en qué posición**

Entre A y C



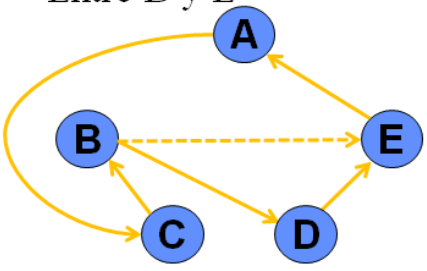

**CTO TOUR = 6 - 1 + (8 + 8) = 21**

Entre C y B



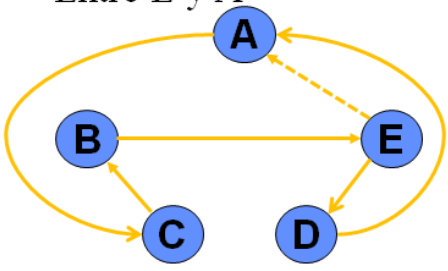

**CTO TOUR = 6 - 2 + (8 + 6) = 18**

Entre B y E



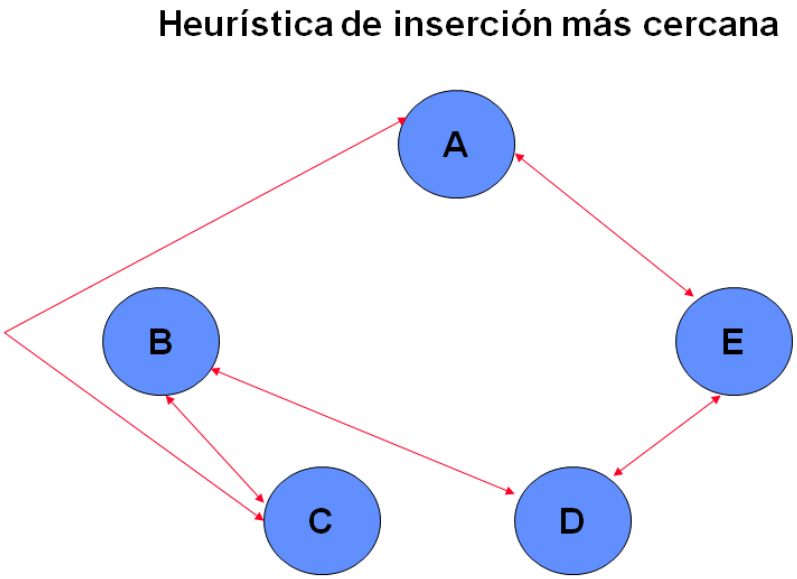

**CTO TOUR = 6 - 1 + (6 + 4) = 15**

Entre E y A




**CTO TOUR = 6 - 2 + (4 + 8) = 16**

Finalmente, esta es la solución que queda:




Esta solución tiene costo igual a 15, es la más barata de las que hemos obtenido con heurísticas.

***<u>Heurística del árbol generador mínimo (para aplicarla se debe cumplir la desigualdad triangular):</u>***

***Representar la posición de las ciudades en un plano x y.***

***Armar el árbol de costo mínimo. Cada ciudad se conecta a la más cercana constituyendo una rama, todas las ramas deben estar conectadas formando el árbol.***

***Recorrer el árbol en profundidad (un tour que visita todas las ciudades y puede pasar más de una vez por cada ciudad.***

***Repetir el recorrido salteando las ciudades ya visitadas.***

## ***Heurística de Barrido:***

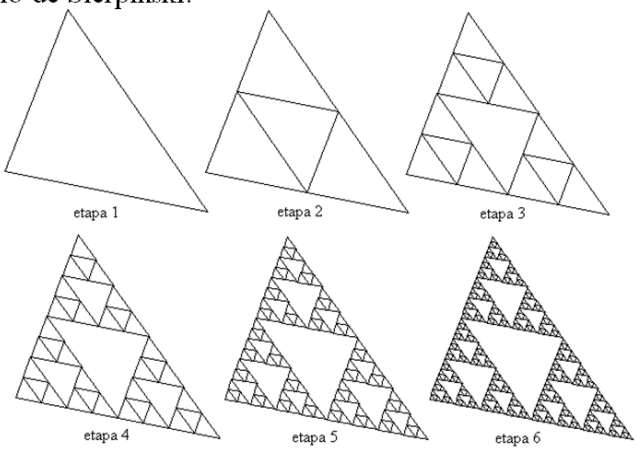
***Se rota una semirecta alrededor del centro del plano y se ponen las ciudades en el tour, a medida que la semirecta las toca.***

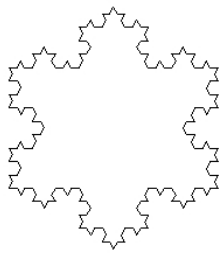
## ***Heurística de Curvas de Sierpinski (L.Platzman y J. Bartholdi´s):***

***Se genera una curva de Sierpinski (también llamada "triángulo de Sierpinski") y se arma el tour en el orden en que la curva va tocando las ciudades.***

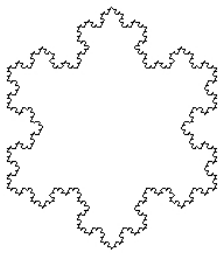
```
static long sierp_pt2code( double ax, double ay, double bx, double by, double cx, double cy,
    int currentLevel, int maxLevel, long code, double x, double y )
{
    if (currentLevel <= maxLevel) {
        currentLevel++;
        if ((sqr(x-ax) + sqr(y-ay)) < (sqr(x-cx) + sqr(y-cy))) {
            code = sierp_pt2code( ax, ay, (ax+cx)/2.0, (ay+cy)/2.0, bx, by,
                currentLevel, maxLevel, 2 * code + 0, x, y );
        }
        else {
            code = sierp_pt2code( bx, by, (ax+cx)/2.0, (ay+cy)/2.0, cx, cy,
                currentLevel, maxLevel, 2 * code + 1, x, y );
        }
    }
    return code;
}
```

Triángulo de Sierpinski:

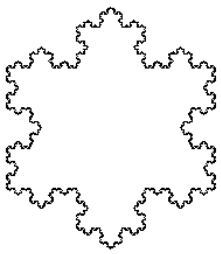




Curva de Koch
de orden 3



Curva de Koch
de orden 4



Curva de Koch
de orden 5

# Heurísticas de Mejoramiento

***K-intercambios:***

***Dada una solución factible (un tour), tratar de mejorarla cambiando al mismo tiempo k ejes de la solución. (HAY QUE EXPLICITAR COMO SE ELIGEN LOS EJES A CAMBIAR)***

***Este procedimiento se basa en una propiedad de grafos euclideos: "Si un ciclo Hamiltoniano se cruza a sí mismo, puede ser fácilmente acortado, basta con eliminar las dos aristas que se cruzan y reconectar los dos caminos resultantes mediante aristas que no se corten. El ciclo final es más corto que el inicial."***

***K-intercambios:***

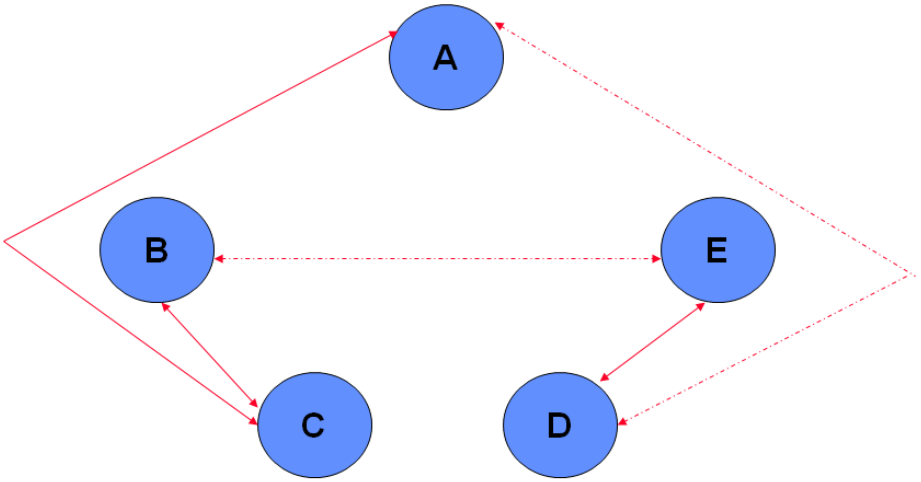
***Técnicas de optimización local:***

- ***Sea S la solución actual.***
- ***Sea S' la solución obtenida al hacer un k-intercambio.***
- ***Si S' es mejor que S, definir S = S'.***
- ***Sino, determinar si existen otros k-intercambios que aún no fueron examinados. Si hay, ir a 1, sino FIN.***

***Las más usadas son 2 intercambios (como 2-Opt)***

***Sin embargo, en las implementaciones nunca se buscan TODOS los posibles intercambios (se coloca un parámetro de tiempo y se busca durante determinado tiempo, si pasado ese tiempo no se mejoró, se queda con la mejor solución encontrada hasta el momento).***

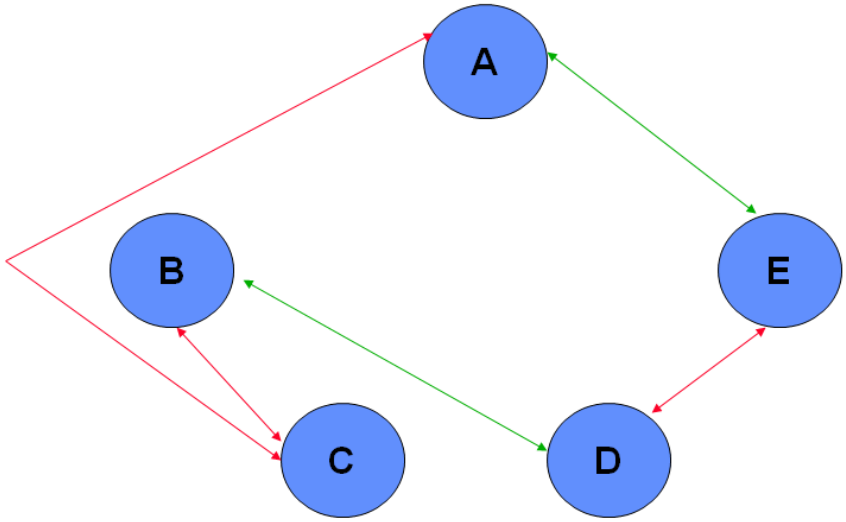
Heurística de mejoramiento de 2-intercambio




Costo total = 16

***Partimos de esta solución eliminando los dos ejes que están punteados y le aplicamos un 2 intercambio para pasar a otra solución, luego verificamos si su funcional es mejor que la solución actual.***

Heurística de mejoramiento de 2-intercambio




Costo total = 15

El funcional es mejor, reemplazando las líneas punteadas por las verdes, por lo que seguiremos probando para mejorar esta solución (olvidamos la anterior)

Para seguir probando partimos de esta solución que acabamos de encontrar.

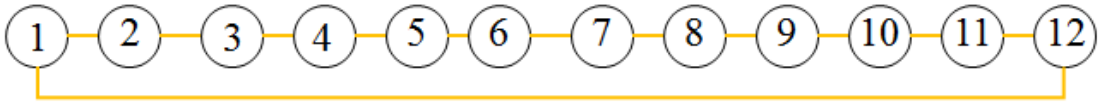
## Heurísticas de Mejoramiento

***Algoritmo de Lin y Kernighan:***
***Considerar un tour inicial***
***marca = 1***
***Mientras marca = 1***
***marca = 0***
***Etiquetar todos los nodos como no explorados***
***Mientras queden nodos sin explorar:***
***Seleccionar un nodo i no explorado aún***
***Examinar todos los movimientos (2-opt, inserción) que incluyan la arista de i a su sucesor en el tour***
**Si alguno de los movimientos reduce la longitud del tour, realizar el mejor de todos y hacer marca = 1**
**Sino, etiquetar el nodo i como explorado**

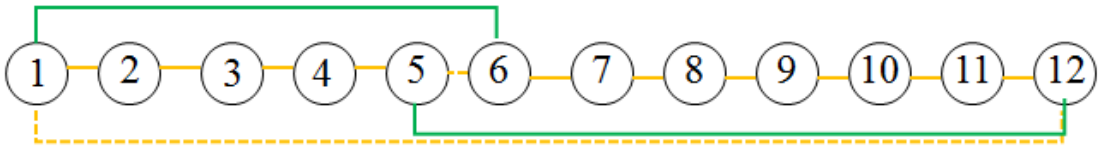
- ***Dado el gran número de combinaciones posibles para escoger los movimientos, es evidente que una implementación eficiente del algoritmo tendrá que restringir el conjunto de movimientos a examinar en cada paso.***
- ***El algoritmo de Lin y Kernighan propone un movimiento compuesto, en donde cada una de las partes consta de un movimiento que no mejora necesariamente pero el movimiento compuesto sí es de mejora. De esta forma es como si se realizaran varios movimientos simples consecutivos en donde algunos empeoran y otros mejoran el valor de la solución, pero no se pierde el control sobre el proceso de búsqueda ya que el movimiento completo mejora.***

Veamos con un ejemplo de viajante de doce ciudades cómo implementar el algoritmo.

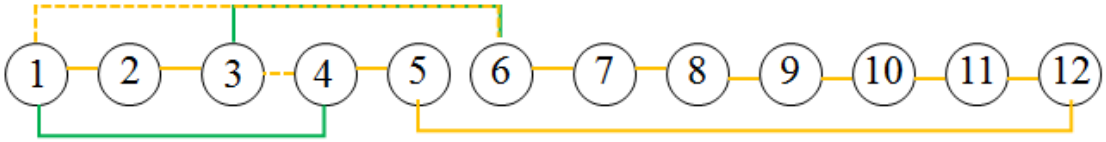
Consideramos el tour inicial dado por el orden natural de los vértices y lo representamos así:



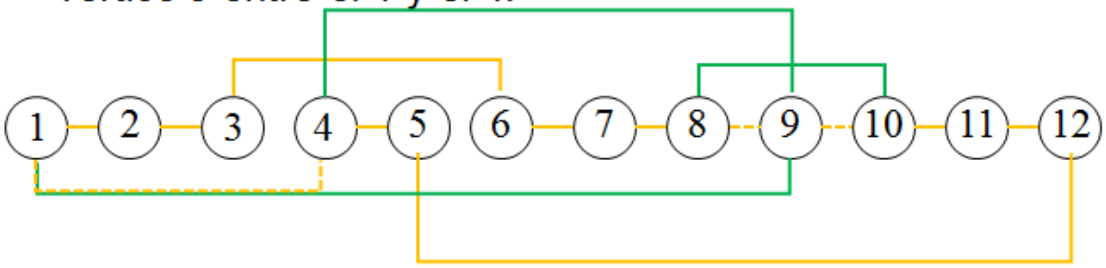

**Paso 1**: Realiza un *movimiento 2-opt* reemplazando las aristas (12,1) y (5,6) por (12,5) y (1,6).




**Paso 2**: Realiza un *movimiento 2-opt* reemplazando las aristas (6,1) y (3,4) por (6,3) y (1,4).




**Paso 3**: Realiza un *movimiento de inserción,* insertando el vértice 9 entre el 1 y el 4.




Una vez en esta solución se puede seguir tratando de mejorar la solución que acabamos de obtener, que es peor que la mejor que teníamos hasta el momento pero que nos permite tratar de "escapar" de óptimos locales.

Más información en:
http://www.math.uwaterloo.ca/tsp/
http://www.math.uwaterloo.ca/tsp/optimal/index.html