Universidad Autónoma de Nuevo León

Facultad de Ciencias Físico Matemáticas

Alumna: **Mayra Alejandra Rivera Rodríguez**

Matricula:**1742899**

Matemáticas Computacionales

Reporte de Problema del agente viajero, vecino más cercano, kruskal y método exacto.

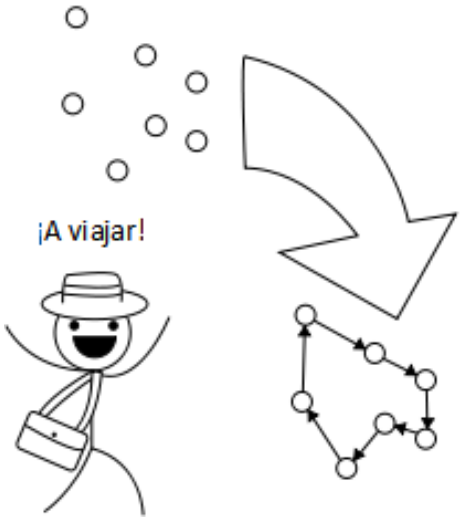
En este reporte se definirá el problema del agente viajero y también de uno de sus métodos de solución que es la del vecino más cercano.

Además de esto dará un ejemplo del mimo aplicado en un problema de elección propia.

1-El problema del agente viajero

Este problema es considerado como un conjunto de grafos cuyas aristas son vistas como las rutas a recorrer para visitar a todos los nodos.

El objetivo que nos plantea este problema es encontrar un recorrido que conecte todos los nodos de un conjunto de modo que los visite solo una vez, regresando a su nodo de inicio y además de esto minimice la distancia que se tiene que recorrer.

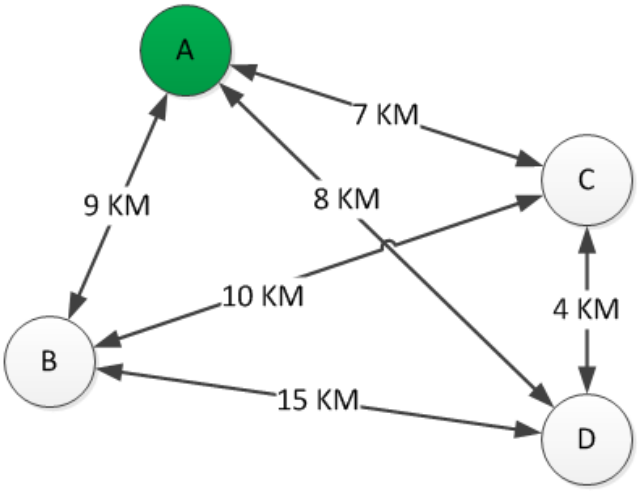


El PAV se clasifica como un problema de optimización combinatoria, es decir, es un problema en el que intervienen cierto número de variables donde cada variable puede tener n valores diferentes por el cual se hacen combinaciones entre ellos y esto da ligar a múltiples soluciones que se calculan en un tiempo finito.

Se considera un problema complicado de resolver por qué no se puede garantizar que se encontrara la mejor solución en un tiempo razonable para los sistemas de cómputo.

Lo más difícil de este problema se podría decir que es la variación que existe en cuanto a las distancias ya que el número de rutas dependerá si las rutas son simétricas o no.

La cantidad de rutas posibles están determinadas por la ecuación (n-1)! Si son simétricas y si no lo fueran se reduciría a la mitad es decir (n-1)! /2.



La complejidad del cálculo del problema del agente viajero ha despertado múltiples iniciativas para mejorar la eficiencia del cálculo de rutas.

Uno de los métodos más básicos que se conoce lleva el nombre de fuerza bruta y este consiste en calcular todos los recorridos posibles, lo cual es muy ineficiente; aunque también existen otros heurísticos que se han desarrollado para facilitar el cálculo de las soluciones posibles al problema algunos ejemplos de ellos son métodos como el del vecino más cercano, la inserción más barata y el doble sentido.

A continuación, se hablara más sobre el método del vecino más cercano y además daremos un ejemplo.

2-Vecino más cercano

Esta heurística se basa en la idea de moverse de un nodo al siguiente, de tal forma que, de todas las opciones, el nodo elegido sea el más cercano a donde se encuentra el agente viajero.

Los pasos que sigue el algoritmo son:

1°Eleccion de un nodo cualquiera con respecto al nodo inicial.

2°Descubrir una distancia entre los nodos(arista) que se la menor a un nodo no visitado

3°visitar el nodo

4°marcarlo como visitado

5°repetir los pasos 2,3,4

6° si todos los nodos han sido visitados cerrar el algoritmo y devolver solución

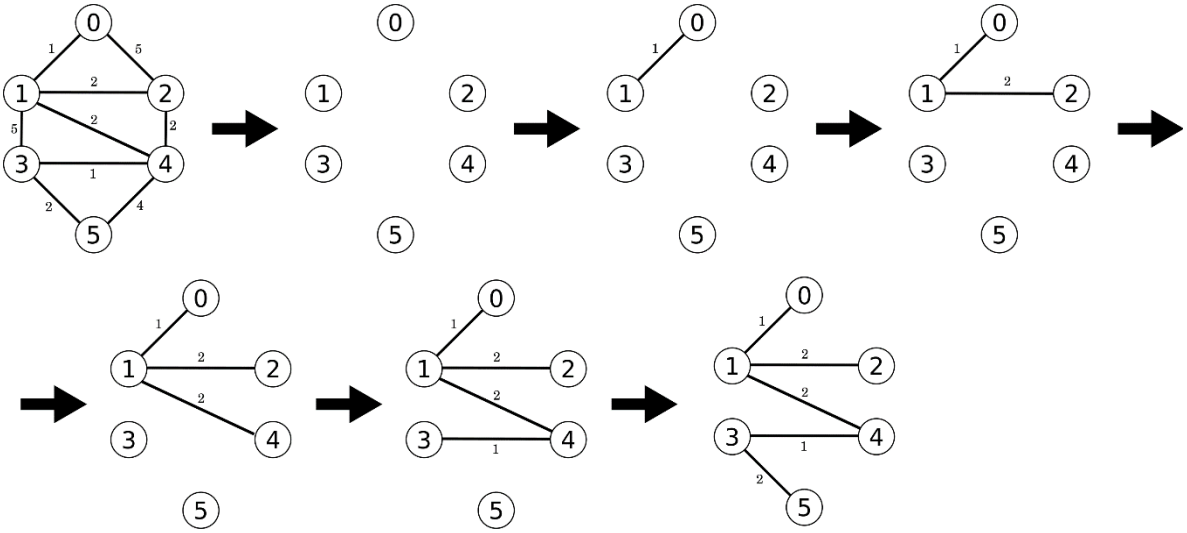
Este método genera un camino corto que puede ser una solución, pero no siempre es la ideal. El algoritmo es fácil de implementar y ejecuta rápidamente, pero algunas de las veces por tomar el primer vecino no se considera el camino que está más adelante y esto hace que se puedan perder rutas más cortas.

3-Algoritmo de aproximación:

Este es un algoritmo que se utiliza para encontrar soluciones aproximadas a problemas de optimización (un ejemplo de problema de optimización es el PAV).

A diferencia de las heurísticas, que usualmente solo encuentran soluciones buenas en tiempos rápidos, lo que busca un algoritmo de aproximación es encontrar soluciones que sean las mejores y que no se tarde tanto.

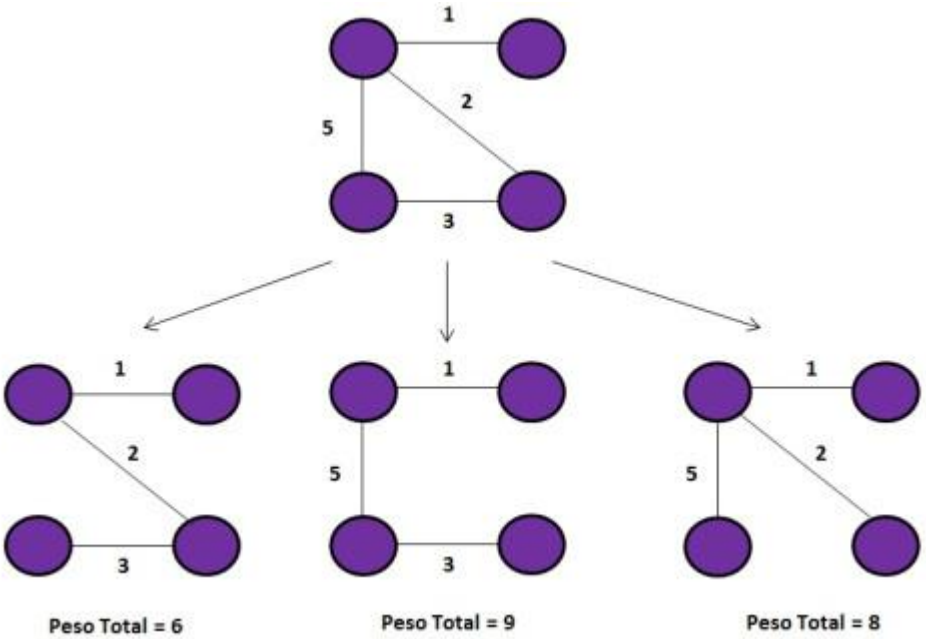
La aproximación mejora la calidad de las soluciones para factores constantes pequeños. Este algoritmo resuelve el problema de manera polinomial pero no óptima.



4- Árbol de Expansión mínima

Dado un grafo, no dirigido y con distancias en las aristas; un árbol de expansión mínima es un árbol compuesto por todos los vértices y cuya suma de las aristas es la de menor cantidad.

Se muestra un ejemplo en la imagen siguiente.



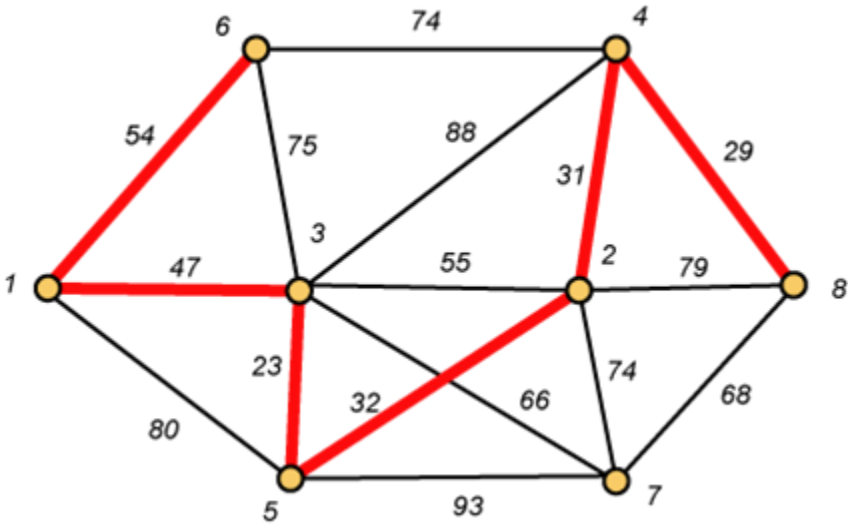
El problema para encontrar un árbol de expansión mínima se puede resolver con varios algoritmos los más conocidos son el Prim y el kruskal ambos usan técnicas voraces (greedy).

A continuación, se hablará del algoritmo kruskal.

5- Algoritmo Kruskal

El objetivo de este algoritmo es construir un árbol formado por aristas sucesivamente seleccionadas de mínima distancia a partir de un grafo con distancias en las aristas y de esta manera llegar a formar un árbol de expansión mínima.

Este algoritmo tiene una complejidad de **m (log m).**



Después de definir todo lo anterior pondremos un problema que se resolverá con el algoritmo de kruskal y todo lo antes ya definido.

Problema de ejemplo

Una distribuidora de muebles que cuanta con varias sucursales en algunos estados de la Republica de entre los cuales las más importantes se encuentran en: Nuevo León, Guerrero, Zacatecas, SLP, Chihuahua, Tamaulipas, Sinaloa, Baja California, Chiapas, Querétaro. Si la empresa quiere distribuir su nuevo modelo de muebles a todas sus sucursales. ¿Cuál sería la ruta deberíamos tomar de tal manera que nos haga recorrer menos distancia y de esta manera ahorrar en Diesel?

Información para la solución del problema

Mapa con Sucursales

Distancia que existe entre los estados donde se encuentran las sucursales (en kilómetros)

	NL	SLP	TAM	CHIA	CHIHU	BC	ZAC	GUE	SIN	QUER
NL	0	564	286	1773	852	1607	511	1306	1141	756
SLP		0	416	1230	1018	1719	194	764	994	208
TAM			0	1327	1107	1778	600	1160	1397	610
CHIA				0	2249	2738	1423	1140	2110	1038
CHIHU					0	923	832	1786	1191	1233
BC						0	1547	2182	993	1876
ZAC							0	962	806	412
GUE								0	1612	562
SIN									0	1117
QUER										0

Para encontrar la solución a este problema se utilizarán tres métodos: el método de kruskal, además se aplicará la heurística del vecino más cercano y un método más exacto

El código que utilizaremos es el siguiente:

```
from heapq import heappop, heappush
from copy import deepcopy
import random

import time
def permutation(lst):
    if len(lst) == 0:
        return []
    if len(lst) == 1:
        return [lst]
    l = [] # empty list that will store current permutation
    for i in range(len(lst)):
       m = lst[i]
       remLst = lst[:i] + lst[i+1:]
       for p in permutation(remLst):
           l.append([m] + p)
    return l

class Fila:
    def __init__(self):
        self.fila= []
    def obtener(self):
        return self.fila.pop()
    def meter(self,e):
        self.fila.insert(0,e)
        return len(self.fila)
    @property
    def longitud(self):
```

```
        return len(self.fila)

class Pila:
    def __init__(self):
        self.pila= []
    def obtener(self):
        return self.pila.pop()
    def meter(self,e):
        self.pila.append(e)
        return len(self.pila)
    @property
    def longitud(self):
        return len(self.pila)



def flatten(L):
    while len(L) > 0:
        yield L[0]
        L = L[1]

class Grafo:

    def __init__(self):
        self.V = set() # un conjunto
        self.E = dict() # un mapeo de pesos de aristas
        self.vecinos = dict() # un mapeo

    def agrega(self, v):
```

```
    self.V.add(v)
    if not v in self.vecinos: # vecindad de v
        self.vecinos[v] = set() # inicialmente no tiene nada

def conecta(self, v, u, peso=1):
    self.agrega(v)
    self.agrega(u)
    self.E[(v, u)] = self.E[(u, v)] = peso # en ambos sentidos
    self.vecinos[v].add(u)
    self.vecinos[u].add(v)

def complemento(self):
    comp= Grafo()
    for v in self.V:
        for w in self.V:
            if v != w and (v, w) not in self.E:
                comp.conecta(v, w, 1)
    return comp

def BFS(self,ni):
    visitados =[]
    f=Fila()
    f.meter(ni)
    while(f.longitud>0):
        na = f.obtener()
        visitados.append(na)
        ln = self.vecinos[na]
        for nodo in ln:
```

```
            if nodo not in visitados:
                f.meter(nodo)
        return visitados


    def DFS(self,ni):
        visitados =[]
        f=Pila()
        f.meter(ni)
        while(f.longitud>0):
            na = f.obtener()
            visitados.append(na)
            ln = self.vecinos[na]
            for nodo in ln:
                if nodo not in visitados:
                    f.meter(nodo)
        return visitados


    def shortest(self, v): # Dijkstra's algorithm
        q = [(0, v, ())] # arreglo "q" de las "Tuplas" de lo que se va a almacenar dondo
0 es la distancia, v el nodo y () el "camino" hacia el
        dist = dict() #diccionario de distancias
        visited = set() #Conjunto de visitados
        while len(q) > 0: #mientras exista un nodo pendiente
            (l, u, p) = heappop(q) # Se toma la tupla con la distancia menor
            if u not in visited: # si no lo hemos visitado
                visited.add(u) #se agrega a visitados
                dist[u] = (l,u,list(flatten(p))[::-1] + [u])           #agrega al diccionario
            p = (u, p) #Tupla del nodo y el camino
            for n in self.vecinos[u]: #Para cada hijo del nodo actual
```

```
            if n not in visited: #si no lo hemos visitado
                el = self.E[(u,n)] #se toma la distancia del nodo acutal hacia el nodo hijo
                heappush(q, (l + el, n, p)) #Se agrega al arreglo "q" la distancia actual mas la ditanacia hacia el nodo hijo, el nodo hijo n hacia donde se va, y el camino
        return dist #regresa el diccionario de distancias


    def kruskal(self):
        e = deepcopy(self.E)
        arbol = Grafo()
        peso = 0
        comp = dict()
        t = sorted(e.keys(), key = lambda k: e[k], reverse=True)
        nuevo = set()
        while len(t) > 0 and len(nuevo) < len(self.V):
            #print(len(t))
            arista = t.pop()
            w = e[arista]
            del e[arista]
            (u,v) = arista
            c = comp.get(v, {v})
            if u not in c:
                #print('u ',u, 'v ',v ,'c ', c)
                arbol.conecta(u,v,w)
                peso += w
                nuevo = c.union(comp.get(u,{u}))
                for i in nuevo:
                    comp[i]= nuevo
        print('MST con peso', peso, ':', nuevo, '\n', arbol.E)
        return arbol
```

```
    def vecinoMasCercano(self):
        ni = random.choice(list(self.V))
        result=[ni]
        while len(result) < len(self.V):
            ln = set(self.vecinos[ni])
            le = dict()
            res =(ln-set(result))
            for nv in res:
                le[nv]=self.E[(ni,nv)]
            menor = min(le, key=le.get)
            result.append(menor)
            ni=menor
        return result
```

```
g= Grafo()
g.conecta('NL','TAM',286)
g.conecta('NL','SLP',564)
g.conecta('NL','CHIA',1773)
g.conecta('NL','CHIHU',852)
g.conecta('NL','BC',1607)
g.conecta('NL','ZAC',511)
g.conecta('NL','GUE',1306)
g.conecta('NL','SIN',1141)
```

```
g.conecta('NL','QUER',756)
g.conecta('SLP','TAM',416)
g.conecta('SLP','CHIA',1230)
g.conecta('SLP','CHIHU',1018)
g.conecta('SLP','BC',1719)
g.conecta('SLP','ZAC',194)
g.conecta('SLP','GUE',764)
g.conecta('SLP','SIN',994)
g.conecta('SLP','QUER',208)
g.conecta('TAM','CHIA',1327)
g.conecta('TAM','CHIHU',1107)
g.conecta('TAM','BC',1778)
g.conecta('TAM','ZAC',600)
g.conecta('TAM','GUE',1160)
g.conecta('TAM','SIN',1397)
g.conecta('TAM','QUER',610)
g.conecta('CHIA','CHIHU',2249)
g.conecta('CHIA','BC',2738)
g.conecta('CHIA','ZAC',1423)
g.conecta('CHIA','GUE',1140)
g.conecta('CHIA','SIN',2110)
g.conecta('CHIA','QUER',1038)
g.conecta('CHIHU','BC',923)
g.conecta('CHIHU','ZAC',832)
g.conecta('CHIHU','GUE',1786)
g.conecta('CHIHU','SIN',1191)
g.conecta('CHIHU','QUER',1233)
g.conecta('BC','ZAC',1547)
```

```
g.conecta('BC','GUE',2182)
g.conecta('BC','SIN',993)
g.conecta('BC','QUER',1876)
g.conecta('ZAC','GUE',962)
g.conecta('ZAC','SIN',806)
g.conecta('ZAC','QUER',412)
g.conecta('GUE','SIN',1612)
g.conecta('GUE','QUER',562)
g.conecta('SIN','QUER',1117)

print(g.kruskal())
#print(g.shortest('c'))



#print(g)
k = g.kruskal()
print([print(x, k.E[x]) for x in k.E])

for r in range(10):
    ni = random.choice(list(k.V))
    dfs =  k.DFS(ni)
    c = 0
    #print(dfs)
    #print(len(dfs))
    for f in range(len(dfs) -1):
            c += g.E[(dfs[f],dfs[f+1])]
            print(dfs[f], dfs[f+1], g.E[(dfs[f],dfs[f+1])] )
```

```
    c += g.E[(dfs[-1],dfs[0])]
    print(dfs[-1], dfs[0], g.E[(dfs[-1],dfs[0])])
    print('costo',c)

vmc = g.vecinoMasCercano()
print(vmc)
c=0
for f in range(len(vmc) -1):
    c += g.E[(vmc[f],vmc[f+1])]
    print(vmc[f], vmc[f+1], g.E[(vmc[f],vmc[f+1])] )

c += g.E[(vmc[-1],vmc[0])]
print(vmc[-1], vmc[0], g.E[(vmc[-1],vmc[0])])
print('vmc costo',c)



data =['NL','SLP','TAM','CHIA','CHIHU','BC','ZAC','GUE','SIN','QUER']
tim=time.clock()
per = permutation(data)
vm, rm= 100000000000,[]
for e in per:
    #print(e)
    c=0
    for f in range(len(e) -1):
        c += g.E[(e[f],e[f+1])]
        #print(e[f], e[f+1], g.E[(e[f],e[f+1])] )

    c += g.E[(e[-1],e[0])]
```

```
    #print(e[-1], e[0], g.E[(e[-1],e[0])])
    if c < vm:
        vm,rm= c,e
    #print('e costo',c)
print(time.clock()-tim)
print('minimo exacto',vm,rm)
```

Los métodos anteriores nos arrojan los siguientes resultados:

Método de kruskal:

°árbol de expansión mínima:

```
MST con peso 5265 : {'CHIA', 'CHIHU', 'SLP', 'TAM', 'QUER', 'NL', 'ZAC'
, 'BC', 'GUE', 'SIN'}
 {('ZAC', 'SLP'): 194, ('SLP', 'ZAC'): 194, ('QUER', 'SLP'): 208, ('SLP
', 'QUER'): 208, ('TAM', 'NL'): 286, ('NL', 'TAM'): 286, ('TAM', 'SLP')
: 416, ('SLP', 'TAM'): 416, ('QUER', 'GUE'): 562, ('GUE', 'QUER'): 562,
('SIN', 'ZAC'): 806, ('ZAC', 'SIN'): 806, ('ZAC', 'CHIHU'): 832, ('CHIH
U', 'ZAC'): 832, ('BC', 'CHIHU'): 923, ('CHIHU', 'BC'): 923, ('QUER', '
CHIA'): 1038, ('CHIA', 'QUER'): 1038}
<__main__.Grafo object at 0x000000EBACE9F4A8>
MST con peso 5265 : {'CHIA', 'CHIHU', 'SLP', 'TAM', 'QUER', 'NL', 'ZAC'
, 'BC', 'GUE', 'SIN'}
 {('ZAC', 'SLP'): 194, ('SLP', 'ZAC'): 194, ('QUER', 'SLP'): 208, ('SLP
', 'QUER'): 208, ('TAM', 'NL'): 286, ('NL', 'TAM'): 286, ('TAM', 'SLP')
: 416, ('SLP', 'TAM'): 416, ('QUER', 'GUE'): 562, ('GUE', 'QUER'): 562,
('SIN', 'ZAC'): 806, ('ZAC', 'SIN'): 806, ('ZAC', 'CHIHU'): 832, ('CHIH
U', 'ZAC'): 832, ('BC', 'CHIHU'): 923, ('CHIHU', 'BC'): 923, ('QUER', '
CHIA'): 1038, ('CHIA', 'QUER'): 1038}
('ZAC', 'SLP') 194
('SLP', 'ZAC') 194
('QUER', 'SLP') 208
('SLP', 'QUER') 208
('TAM', 'NL') 286
('NL', 'TAM') 286
('TAM', 'SLP') 416
('SLP', 'TAM') 416
('QUER', 'GUE') 562
('GUE', 'QUER') 562
('SIN', 'ZAC') 806
('ZAC', 'SIN') 806
('ZAC', 'CHIHU') 832
('CHIHU', 'ZAC') 832
('BC', 'CHIHU') 923
('CHIHU', 'BC') 923
('QUER', 'CHIA') 1038
('CHIA', 'QUER') 1038
```

La mejor ruta que nos arroja el método de kruskal es :

```
BC CHIHU 923
CHIHU ZAC 832
ZAC SLP 194
SLP TAM 416
TAM NL 286
NL QUER 756
QUER GUE 562
GUE CHIA 1140
CHIA SIN 2110
SIN BC 993
costo 8212
```

Método del vecino más cercano:

Los siguientes resultados que se mostraran serán tomando como nodo inicial (punto de partida) cada uno de los 10 destinos.

```
['CHIHU', 'ZAC', 'SLP', 'QUER', 'GUE', 'CHIA', 'TAM', 'NL', 'SIN', 'BC']
CHIHU ZAC 832
ZAC SLP 194
SLP QUER 208
QUER GUE 562
GUE CHIA 1140
CHIA TAM 1327
TAM NL 286
NL SIN 1141
SIN BC 993
BC CHIHU 923
vmc costo 7606

['SLP', 'ZAC', 'QUER', 'GUE', 'CHIA', 'TAM', 'NL', 'CHIHU', 'BC', 'SIN']
SLP ZAC 194
ZAC QUER 412
QUER GUE 562
GUE CHIA 1140
CHIA TAM 1327
TAM NL 286
NL CHIHU 852
CHIHU BC 923
BC SIN 993
SIN SLP 994
vmc costo 7683
```

```
['GUE', 'QUER', 'SLP', 'ZAC', 'NL', 'TAM', 'CHIHU', 'BC', 'SIN', 'CHIA']
GUE QUER 562
QUER SLP 208
SLP ZAC 194
ZAC NL 511
NL TAM 286
TAM CHIHU 1107
CHIHU BC 923
BC SIN 993
SIN CHIA 2110
CHIA GUE 1140
vmc costo 8034

['BC', 'CHIHU', 'ZAC', 'SLP', 'QUER', 'GUE', 'CHIA', 'TAM', 'NL', 'SIN']
BC CHIHU 923
CHIHU ZAC 832
ZAC SLP 194
SLP QUER 208
QUER GUE 562
GUE CHIA 1140
CHIA TAM 1327
TAM NL 286
NL SIN 1141
SIN BC 993
vmc costo 7606

['CHIA', 'QUER', 'SLP', 'ZAC', 'NL', 'TAM', 'CHIHU', 'BC', 'SIN', 'GUE']
CHIA QUER 1038
QUER SLP 208
SLP ZAC 194
ZAC NL 511
NL TAM 286
TAM CHIHU 1107
CHIHU BC 923
BC SIN 993
SIN GUE 1612
GUE CHIA 1140
vmc costo 8012

['ZAC', 'SLP', 'QUER', 'GUE', 'CHIA', 'TAM', 'NL', 'CHIHU', 'BC', 'SIN']
ZAC SLP 194
SLP QUER 208
QUER GUE 562
GUE CHIA 1140
CHIA TAM 1327
TAM NL 286
NL CHIHU 852
CHIHU BC 923
BC SIN 993
SIN ZAC 806
vmc costo 7291
```

```
['NL', 'TAM', 'SLP', 'ZAC', 'QUER', 'GUE', 'CHIA', 'SIN', 'BC', 'CHIHU']
NL TAM 286
TAM SLP 416
SLP ZAC 194
ZAC QUER 412
QUER GUE 562
GUE CHIA 1140
CHIA SIN 2110
SIN BC 993
BC CHIHU 923
CHIHU NL 852
vmc costo 7888

['TAM', 'NL', 'ZAC', 'SLP', 'QUER', 'GUE', 'CHIA', 'SIN', 'BC', 'CHIHU']
TAM NL 286
NL ZAC 511
ZAC SLP 194
SLP QUER 208
QUER GUE 562
GUE CHIA 1140
CHIA SIN 2110
SIN BC 993
BC CHIHU 923
CHIHU TAM 1107
vmc costo 8034

['QUER', 'SLP', 'ZAC', 'NL', 'TAM', 'CHIHU', 'BC', 'SIN', 'GUE', 'CHIA']
QUER SLP 208
SLP ZAC 194
ZAC NL 511
NL TAM 286
TAM CHIHU 1107
CHIHU BC 923
BC SIN 993
SIN GUE 1612
GUE CHIA 1140
CHIA QUER 1038
vmc costo 8012

['SIN', 'ZAC', 'SLP', 'QUER', 'GUE', 'CHIA', 'TAM', 'NL', 'CHIHU', 'BC']
SIN ZAC 806
ZAC SLP 194
SLP QUER 208
QUER GUE 562
GUE CHIA 1140
CHIA TAM 1327
TAM NL 286
NL CHIHU 852
CHIHU BC 923
BC SIN 993
vmc costo 7291
```

Método exacto:

```
80.67766005489608
minimo exacto 7291 ['NL', 'TAM', 'CHIA', 'GUE', 'QUER', 'SLP', 'ZAC', 'SIN', 'BC', 'CHIHU']
```

Conclusiones

Para encontrar la mejor solución a un PAV como el del ejemplo anterior el mejor método a utilizar para hallar la mejor ruta es el de el método exacto aunque la única cosa que tiene es que lleva mas tiempo calcularlo que los otros dos así que ya seria decisión de cada quien cual método tomar si por rapidez en la realización o por exactitud en cuanto a la ruta.