LABORATORIO DI RICERCA OPERATIVA 2020-2021

Laboratorio OPL - LEZIONE 6

Sommario

1) Il Problema del Cammino di Costo Minimo Vincolato

2) Il Problema del Commesso Viaggiatore su Grafo Orientato (ATSP)

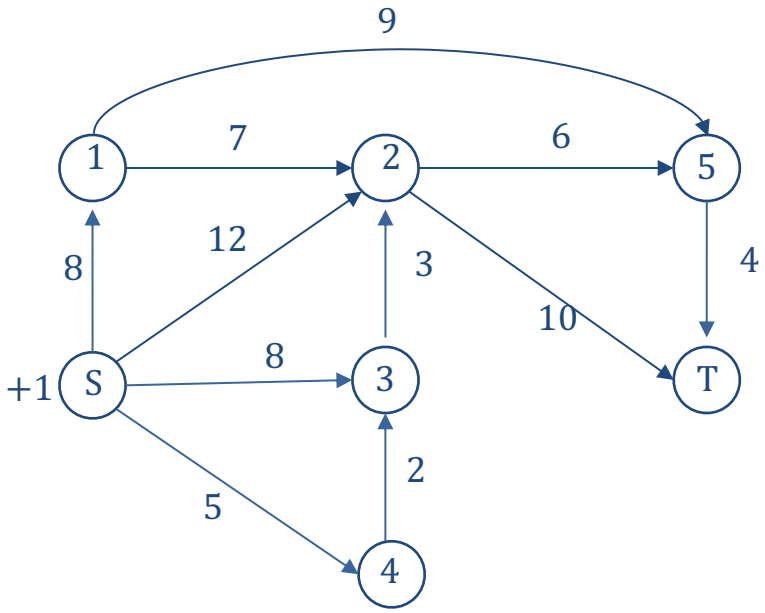
1. Modelli
2. Un po di storia sugli algoritmi risolutivi
3. Applicazioni
4. Implementazione OPL

Cammino di Costo Minimo dal nodo S al nodo T (Lezione 11)

$$\min \sum_{(i,j)\in E} c_{ij} f_{ij}$$

$$\sum_{j|(i,j)\in E} f_{ij} - \sum_{j|(j,i)\in E} f_{ji} = \begin{cases} 1 & i = S \\ 0 & i \neq S,T \\ -1 & i = T \end{cases}$$

$$f_{ij} \geq 0 \quad \forall (i,j) \in E$$



E' un caso particolare del problema di flusso di costo minimo

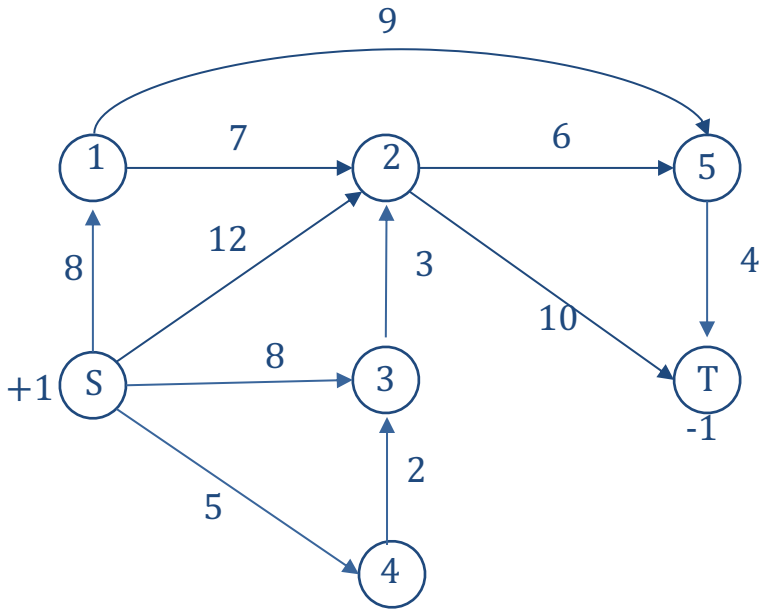
1) Nel grafo è presente un solo nodo sorgente S con divergenza +1 un solo nodo pozzo T con divergenza -1 Tutti i restanti nodi hanno divergenza nulla.

2) Costi sugli archi >= 0

3) Le soluzioni ammissibili di base sono rappresentate da alberi

4) Le variabili di base assumono sempre valore intero (La matrice dei vincoli è la matrice di incidenza di un grafo orientato, che è TUM (Lezione 14))

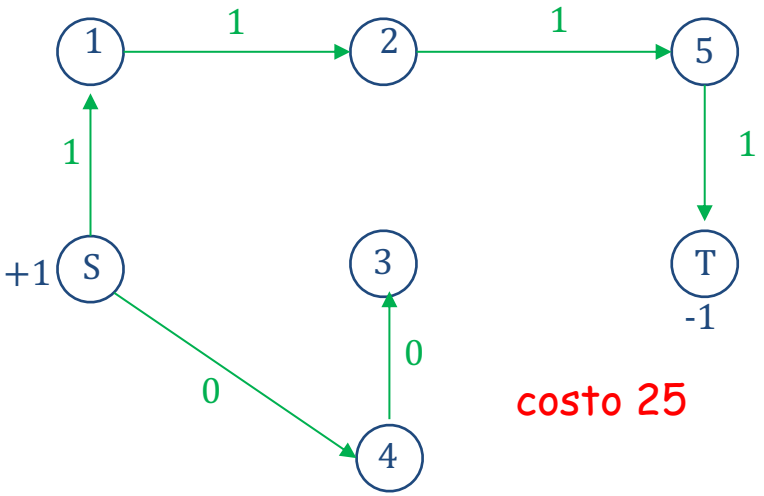
Caso Particolare 4: Cammino di Costo Minimo dal nodo S al nodo T

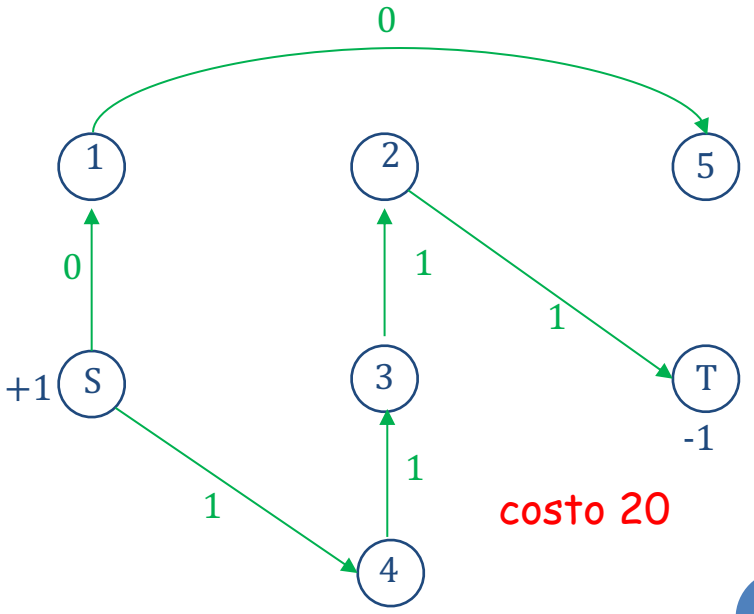


1) In ogni soluzione ammissibile di base l'unica unità di flusso disponibile presso il nodo S raggiungerà il nodo T attraversando archi, senza mai dividersi.

2) In ogni soluzione ammissibile di base, gli archi con flusso non nullo (e pari ad 1), individuano un cammino orientato da S a T.

3) La soluzione ottima rappresenta il cammino di costo minimo da S a T.



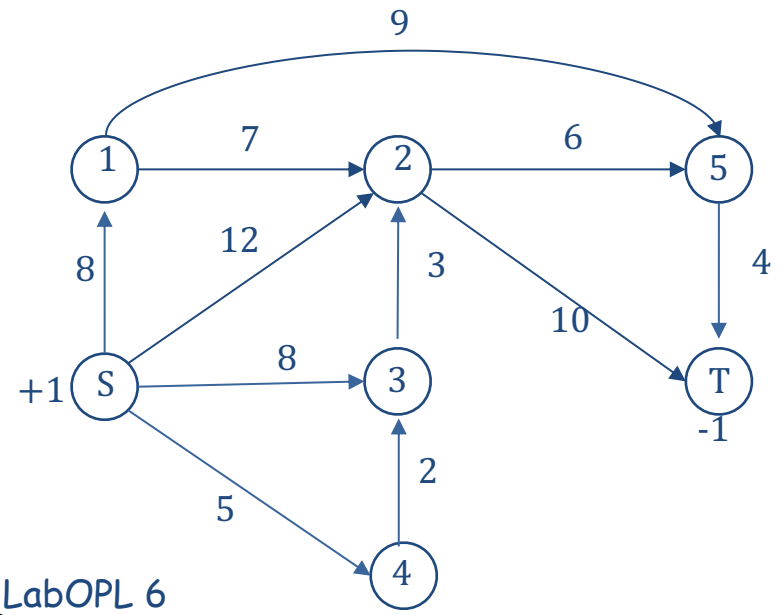


Cammino di Costo Minimo dal nodo S al nodo T vincolato

E' dato un grafo orientato $G =< V, E >$ in cui sono presenti un nodo sorgente (in senso forte) S ed un nodo terminale T (in senso forte).
Il grafo è pesato sugli archi:

1. $c_{ij} \geq 0$: costo necessario ad attraversare l'arco (i, j);
2. $t_{ij} \geq 0$: quantità di una certa «risorsa» (esempio tempo) consumata attraversando l'arco (i, j);

Si vuole determinare su G il cammino di costo minimo da S a T, tale che la quantità totale di risorsa consumata non superi un certo quantitativo B.



Cammino di Costo Minimo dal nodo S al nodo T vincolato

$$min \quad \sum_{(i,j)\in E} c_{ij} f_{ij}$$

$$\sum_{j|(S,j)\in E} f_{Sj} = 1$$

Tra tutti gli archi uscenti da S, uno solo può far parte del cammino

$$\sum_{j|(i,j)\in E} f_{ij} - \sum_{j|(j,i)\in E} f_{ji} = 0 \qquad i \neq S,T$$

Se il cammino tocca il nodo i, uno solo degli archi uscenti da i ed uno solo degli archi entranti in i possono far parte del cammino

$$-\sum_{j|(j,T)\in E} f_{jT} = -1$$

Tra tutti gli archi entranti in T, uno solo può far parte del cammino

$$\sum_{(i,j)\in E} t_{ij} f_{ij} \leq T$$

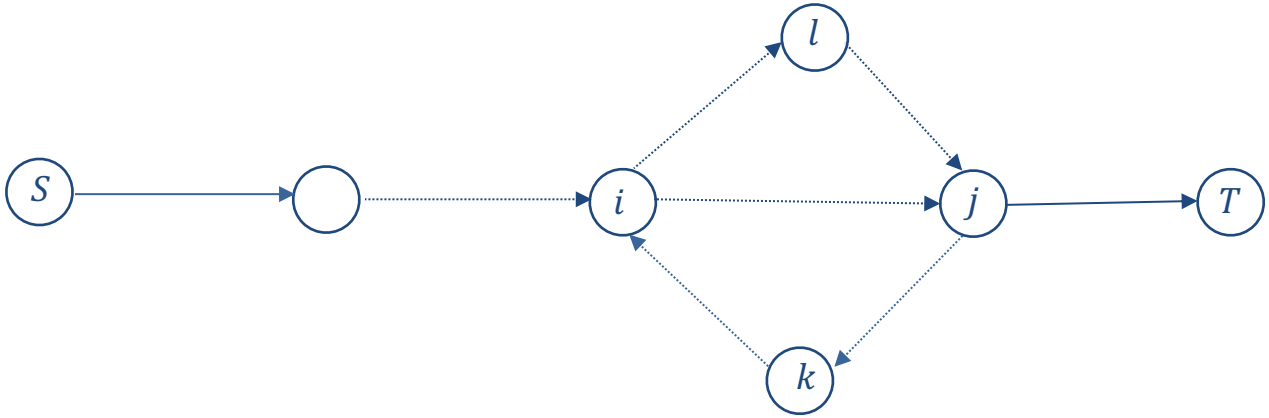
La durata totale del cammino non può superare T

$$f_{ij} \in \{0,1\} \quad (i,j) \in E$$

Cammino di Costo Minimo dal nodo S al nodo T vincolato: Una precisazione

$$\sum_{j|(i,j)\in E} f_{ij} \quad - \quad \sum_{j|(j,i)\in E} f_{ji} \quad = \quad 0 \quad i \neq S,T$$

In un nodo i si deve entrare ed uscire lo stesso numero di volte!



Insieme di archi (variabili decisionali = 1) che soddisfano tutti i vincoli, ma che non rappresentano un cammino elementare (gli archi $(i,l),(l,j),(j,k),(k,i)$ formano un ciclo orientato)!

$$\sum_{\substack{i\in W \\ j\in W}} f_{ij} \leq |W| - 1 \qquad \forall W \subseteq V \setminus \{S,T\}$$

Nel caso generale con costi non necessariamente >=0, occorre aggiungere alla formulazione i «vincoli di eliminazione dei sotto-cicli»

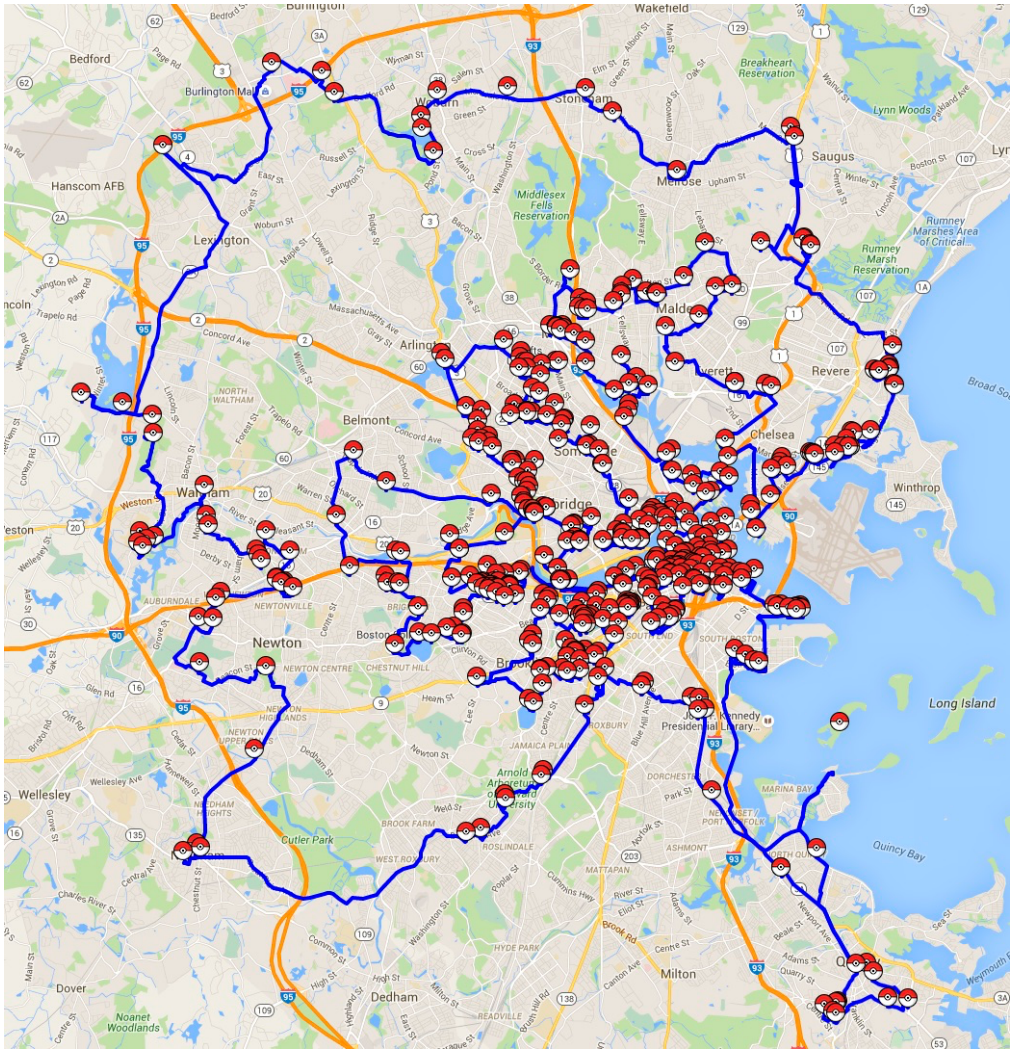
Giochiamo a Pokémon Go?

Da Wikipedia

«Il gioco si svolge in una mappa che riproduce l'ambiente circostante…… Ci sono due categorie principali di luoghi in Pokémon GO: i Pokéstop e le Palestre. I Pokéstop e le Palestre corrispondono a dei luoghi predefiniti del mondo reale, e il giocatore deve stare entro una certa distanza per poter interagire con essi (anche se possono essere controllati se rientrano nel raggio di azione della mappa). L'avatar del protagonista è circondato da un cerchio che si allarga che corrisponde a circa 25 metri: in questo raggio d'azione può interagire con PokéStop e palestre, inoltre può far apparire i pokemon che stanno nascosti in quel raggio. Entrambi i luoghi consentono al giocatore di ottenere oggetti e Uova girando il Disco Foto, ma solo i Pokéstop danno le missioni. Nelle Palestre, i giocatori possono lottare per guadagnare Polvere di Stella e Pokémonete. I Pokéstop sono molto più comuni delle Palestre.
Quando un Pokémon appare si deve fare un tap col dito per entrare nella modalità cattura, che può anche utilizzare la fotocamera del dispositivo per simulare la presenza del Pokémon nel reale ambiente circostante (modalità "AR", augmented reality). ……L'apparizione dei Pokémon avviene su determinati "nidi" (spawn) a intervalli variabili.»

Scelto un qualsiasi Pokestop o Palestra, ci poniamo come obiettivo di visitare tutti i rimanenti posti una ed una volta sola, ma vogliamo camminare il meno possibile

Giochiamo a Pokémon Go a Boston



551 tra Pokéstop e le Palestre

Il tour ottimale è lungo 351003 metri (😥)

Come determinare il tour ottimo?

Il Problema del Commesso Viaggiatore Asimmetrico - ATSP

E' dato un grafo orientato completo $G =< V, E >$ con n nodi $n(n-1)$ archi (sono esclusi cioè gli autoanelli). Un circuito Hamiltoniano τ su G è un percorso chiuso che attraversa tutti i nodi di G una ed una sola volta. Se il grafo è pesato sugli archi con pesi c_{ij} ha senso cercare il circuito Hamiltoniano di peso (costo) minimo.

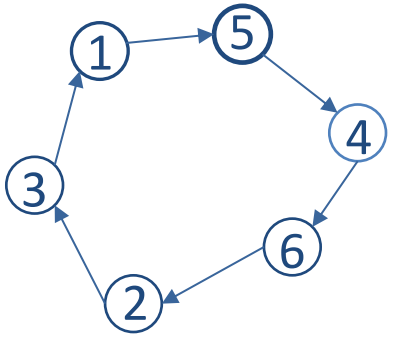
Osservazione:

Un circuito Hamiltoniano su G è un sottografo di $G, \tau =< V, T \subset E >$ tale che

1. $|T| = n = |V|$
2. Fortemente connesso: per ogni coppia di nodi in V esiste un cammino orientato che li congiunge.
3. Al di fuori del circuito Hamiltoniano, non contiene altri sottocicli.

E' facile verificare se vale la proprietà 1, le proprietà 2 e 3 sono equivalenti.

Esempio:
Se G è il grafo completo con 6 nodi, un circuito Hamiltoniano su G è il seguente



ATSP- Formulazione Matematica - 1

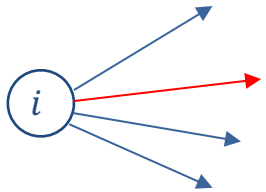
Associamo una variabile decisionale ad ogni arco (i,j) di G

$x_{ij} = 1$ se l'arco (i,j) fa parte di τ; altrimenti $x_{ij} = 0$

Se ogni nodo i di G deve essere attraversato una ed una sola volta, allora in τ

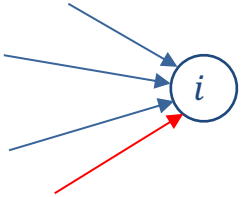
1. sul nodo i deve incidere un solo arco uscente (arco di tipo (i,j))

$$\sum_{\substack{j=1 \\ j\neq i}}^{n} x_{ij} = 1 \qquad \forall i = 1,..n$$



2. sul nodo i deve incidere un solo arco entrante (arco di tipo (j,i))

$$\sum_{\substack{j=1 \\ j\neq i}}^{n} x_{ji} = 1 \qquad \forall i = 1,..n$$



I vincoli 1 e 2 sono detti «vincoli di grado» sugli archi uscenti da i ed entranti in i e fanno in modo che siano scelti esattamente n archi di E (proprietà 1).

ATSP- Formulazione Matematica - 2

Se τ deve essere fortemente connesso, allora

3. Comunque si scelga un sottoinsieme W di V con almeno due nodi, allora deve esistere in τ almeno un arco che congiunga uno dei nodi di W con uno dei nodi che sta in nel complemento di W

$$\sum_{\substack{i\in W \\ j\in \bar{W}}} x_{ij} \geq 1 \qquad \forall W \subset V, \qquad 2 \leq |W| \leq n-2$$

Oppure

4. Comunque si scelga un sottoinsieme W di V con almeno due nodi, in τ possono esistere al più $|W|-1$ archi che incidono su nodi di W.

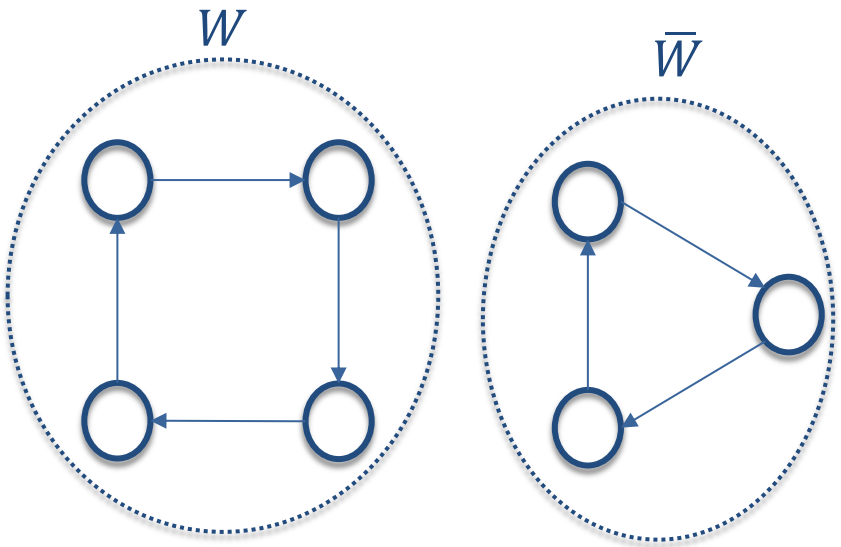
$$\sum_{\substack{i\in W \\ j\in W}} x_{ij} \leq |W|-1 \qquad \forall W \subset V, \qquad 2 \leq |W| \leq n-2$$

I vincoli 3 sono detti «vincoli di connessione o di taglio» (Proprietà 2). I vincoli 4 sono i «vincoli di eliminazione dei sotto-cicli» (Proprietà 3).

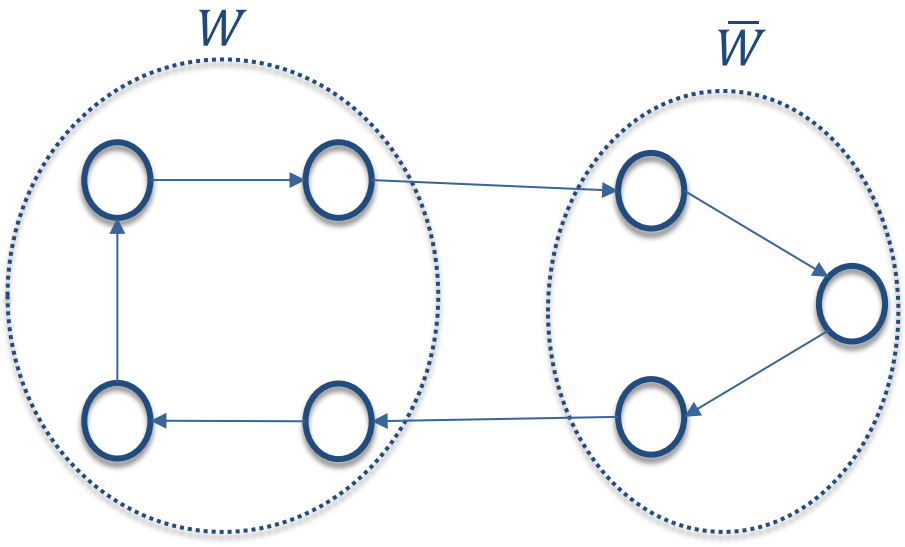
ATSP- Equivalenza fra i vincoli 3 e 4

3) $\sum_{\substack{i \in W \\ j \in \bar{W}}} x_{ij} \geq 1 \qquad \forall W \subset V, \quad 2 \leq |W| \leq n-2$

4) $\sum_{\substack{i \in W \\ j \in W}} x_{ij} \leq |W| - 1 \qquad \forall W \subset V, \quad 2 \leq |W| \leq n-2$



Assegnamento di valori alle variabili che soddisfano 1 e 2, ma non soddisfano 3 e 4



Assegnamento di valori alle variabili che soddisfano 1, 2, 3 e 4

ATSP- Formulazione di Dantzig-Fulkerson-Johnson

$$\min \sum_{i=1}^{n} \sum_{\substack{j=1 \\ j \neq i}}^{n} c_{ij} x_{ij}$$

$$\sum_{\substack{j=1 \\ j \neq i}}^{n} x_{ij} = 1 \qquad \forall i = 1,..n$$

$$\sum_{\substack{j=1 \\ j \neq i}}^{n} x_{ji} = 1 \qquad \forall i = 1,..n$$

$$\sum_{\substack{i \in W \\ j \in \bar{W}}} x_{ij} \geq 1 \qquad \forall W \subset V, \qquad 2 \leq |W| \leq n-2$$

oppure

$$\sum_{\substack{i \in W \\ j \in W}} x_{ij} \leq |W| - 1 \qquad \forall W \subset V, \qquad 2 \leq |W| \leq n-2$$

$$x_{ij} \in \{0,1\} \qquad \forall i,j = 1,..n$$

ATSP- Formulazione di Dantzig-Fulkerson-Johnson

Alcuni dati

Il numero di variabili è dell'ordine di nxn. I vincoli 3 (ma lo stesso dicasi per i vincoli 4) sono scritti per ogni sottoinsieme W con almeno 2 nodi e non più di $n-2$ nodi. Il numero di tali vincoli è, pertanto,

$$2^n - n - n - 1 - 1$$

Cioè pari al numero totale di sottoinsiemi di V esclusi

- ✓ i sottoinsiemi «singleton»
- ✓ i sottoinsiemi di $n-1$ nodi
- ✓ l'insieme vuoto
- ✓ l'insieme V

Ad esempio per

- $n = 10$ # vincoli 1024 - 22=1002
- $n = 20$ # vincoli 1048576 - 42=1048534
- …….
- $n = 120$ #vincoli ≈0.6x10E+36

ATSP – Formulazione di Miller-Zemlin-Tucker

$$\min \sum_{i=1}^{n} \sum_{\substack{j=1 \\ j \neq i}}^{n} c_{ij} x_{ij}$$

$$\sum_{\substack{j=1 \\ j \neq i}}^{n} x_{ij} = 1 \qquad \forall i = 1,..n$$

$$\sum_{\substack{j=1 \\ j \neq i}}^{n} x_{ji} = 1 \qquad \forall i = 1,..n$$

Si introducono $n-1$ ulteriori variabili continue relative ai nodi $2,\dots,n$ e si sostituiscono i vincoli 3 (o 4) con i nuovi vincoli

$$u_i - u_j + n x_{ij} \leq n - 1 \qquad \forall i,j = 2,..,n \quad i \neq j$$

$$x_{ij} \in \{0,1\} \qquad \forall i,j = 1,..,n$$

$$u_i \geq 0 \qquad \forall i = 2,..,n$$

Teorema (Tutto ha un costo!)

MZT è una formulazione debole dell'ATSP rispetto alla formulazione DFJ ⇒ il lower bound fornito dal rilassato lineare di MZT è minore del lower bound fornito dalla formulazione DFJ.

TSP – Storia della Risolubilità

Given a collection of cities and the cost of travel between each pair of them, the traveling salesman problem, or TSP for short, is to find the cheapest way of visiting all of the cities and returning to your starting point. In the standard version we study, the travel costs are symmetric in the sense that traveling from city X to city Y costs just as much as traveling from Y to X.

The simplicity of the statement of the problem is deceptive -- the TSP is one of the most intensely studied problems in computational mathematics and yet no effective solution method is known for the general case. Indeed, the resolution of the TSP would settle the P versus NP problem and fetch a $1,000,000 prize from the Clay Mathematics Institute.

Although the complexity of the TSP is still unknown, for over 50 years its study has led the way to improved solution methods in many areas of mathematical optimization.

http://www.math.uwaterloo.ca/tsp/

TSP – Storia della Risolubilità

Dantzig, Fulkerson, and Johnson showed a way to solve large instances of the TSP; all that came afterward is just icing on the cake (Applegate, Bixby, Chvatal, Cook, 1995)

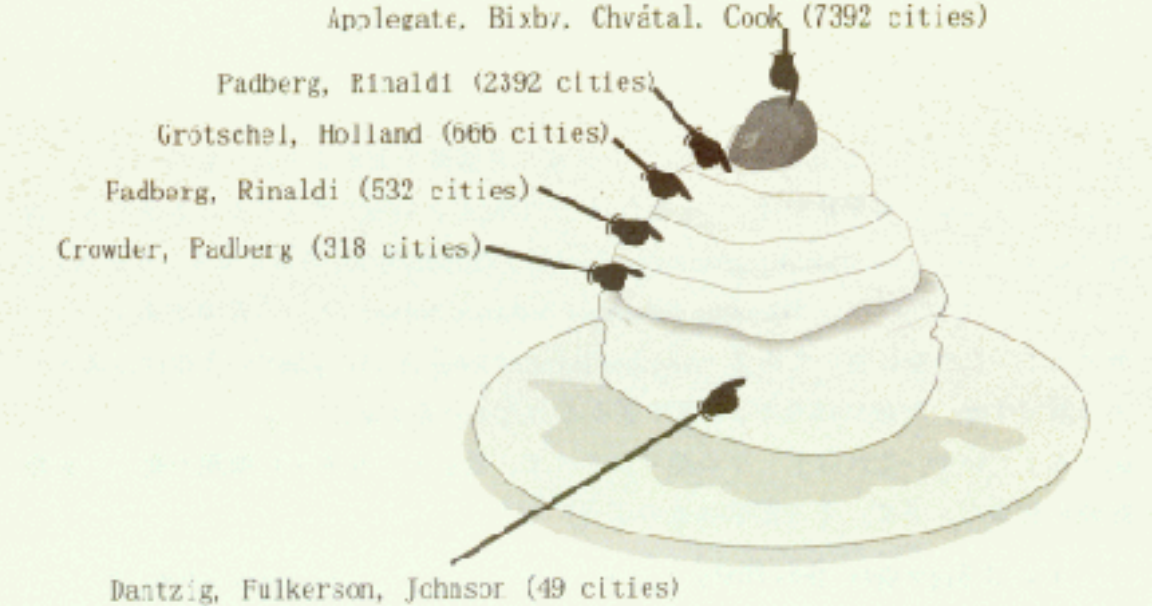


図 6.1 Icing on the cake

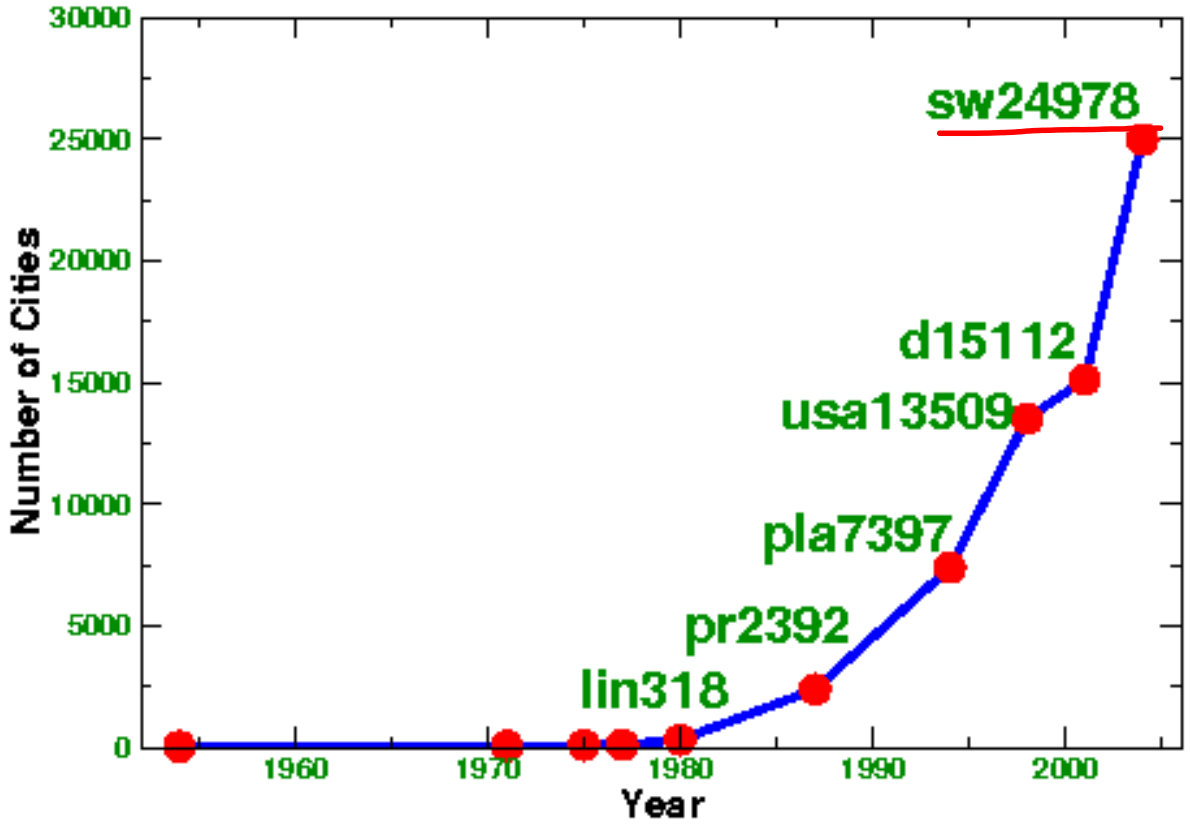
つまりだな，Dantzig, Fulkerson それに Johnson は大規模な巡回セールスマン問題を解く cake を示した．だから，その後にやって来たものはすべて icing on the cake だといっている．

Yoshitsugu Yamamoto and Mikio Kubo "Invitation to the Traveling Salesman Problem"

TSP – Storia della Risolubilità

Crescita esponenziale dovuta a

1. Crescita della potenza di calcolo;
2. Introduzione di nuove tecniche risolutive



TSP – Storia della Risolubilità

Il software CONCORDE

Concorde is a computer code for the symmetric traveling salesman problem (TSP) and some related network optimization problems. The code is written in the ANSI C programming language and it is available for academic research use; for other uses, contact William Cook for licensing options.
Concorde's TSP solver has been used to obtain the optimal solutions to the full set of 110 TSPLIB instances, the largest having 85,900 cities.

http://www.math.uwaterloo.ca/tsp/concorde/

TSP – Records (http://www.math.uwaterloo.ca/tsp/concorde/)

In May 2004, the traveling salesman problem of visiting all 24978 cities in Sweden was solved.

At the time of the computation, this was the largest solved TSP instance, surpassing the previous record of 15112 cities through Germany set in April 2001. The current record an 85900-city tour that arose in a chip-design application.

The majority of the work was carried out on a cluster of 96 dual processor Intel Xeon 2.8 GHz workstations at Georgia Tech's School of Industrial and Systems Engineering, running as a background process when the machines were not otherwise active.

We began the procedure on the first LP in March 2003 and the final branch-and-cut run finished in January 2004. After the run completed, we made a final check of the 14827429 cutting planes (besides subtour inequalities) that were generated and used during the process. This final checking was completed in May 2004.

The cumulative CPU time used in the five branch-and-cut runs and in the cutting-plane procedures for the five root LPs was approximately 84.8 CPU years* on a single Intel Xeon 2.8 GHz processor.

*A CPU Year is simply the amount of computing work done by a 1 GFLOP reference machine in a year of dedicated service (8760 hours).

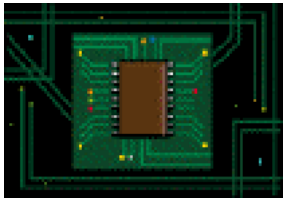
TSP – Current Challenge

To provide a large-scale traveling salesman problem challenge, we put together data from the National Imagery and Mapping Agency database of geographic feature names and data from the Geographic Names Information System (GNIS), to create a 1,904,711-city instance of locations throughout the world. From the data bases, we selected all locations that were registered as populated places, and also included several research bases in Antarctica.

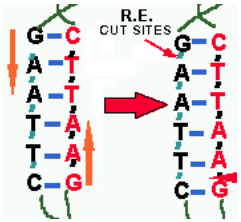
The best (non optimal, with a relative error of about 0.0474%) reported tour for the World TSP was found by by Keld Helsgaun using a variant The tour of length 7,515,772,107 was found on March 13, 2018, improving Helsgaun's previous records of

- ✓ 7,515,772,212 (May 24, 2013),
- ✓ 7,515,778,188 (October 25, 2011),
- ✓ 7,515,786,987 (April 4, 2011),
- ✓ 7,515,796,609 (May 4, 2010),
- ✓ 7,515,877,991 (May 12, 2009),
- ✓ 7,515,947,511 (November 27, 2008),
- ✓ 7,517,285,610 (September 16, 2003).

TSP – TSP non è soltanto un gioco o una sfida!



A semi-conductor manufacturer has used Concorde in experiments to optimize scan chains in integrated circuits. Scan chains are routes included on a chip for testing purposes and it is useful to minimize their length for both timing and power reasons.



A group at AT&T used Concorde to compute DNA sequences. In the application, a collection of DNA strings, each of length k, were embedded in one universal string (that is, each of the target strings is contained as a substring in the universal string), with the goal of minimizing the length of the universal string. The cities of the TSP are the target strings, and the cost of travel is k minus the maximum overlap of the corresponding strings.

Circuit Board Drilling

TSP – ARTE

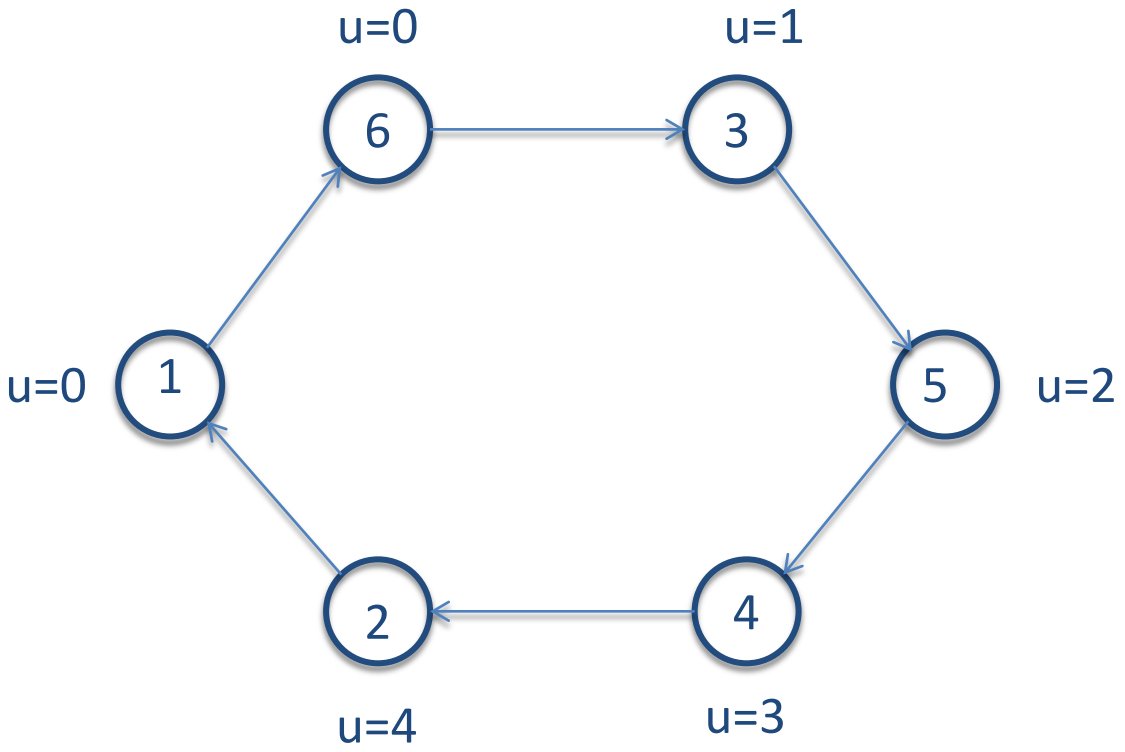
$1000 Prize Offered
In February 2009, Robert Bosch created a 100,000-city instance of the traveling salesman problem (TSP) that provides a representation of Leonardo da Vinci's Mona Lisa as a continuous-line drawing. An optimal solution to the 100,000-city Mona Lisa instance would set a new world record for the TSP. If you have ideas for producing good tours or for showing that a solution is optimal, this is a very nice challenge problem! I would be happy to report any computational results you obtain on this example. The data set in TSPLIB format is given in the file

Formulazione MZT- Implementazione OPL

```
int NNodi = ...;
range Nodi = 1..Nnodi;
int Costo[Nodi][Nodi] = ...;
dvar int X[Nodi][Nodi] in 0..1;
dvar float+ U[Nodi];
minimize sum(i in Nodi, j in Nodi : i != j) Costo[i][j]*X[i][j];
subject to{
  forall(i in Nodi)
    archi_uscenti_da_i:
       sum(j in Nodi : j != i) X[i][j] == 1;
   forall(i in Nodi)
    archi_entranti_in_i:
       sum(j in Nodi : j != i) X[j][i] == 1;
   forall(i in Nodi : i !=1)
     forall(j in Nodi : j != 1 && j != i)
       U[i] - U[j] + NNodi*X[i][j] <= NNodi -1;
  }
tuple coppia{
  int i;
  int j;
 }
 {coppia} Coppie = {<i, j> | i in Nodi, j in Nodi: X[i][j]==1};
execute DISPLAY {
  writeln("Coppie = ", Coppie);
  writeln("Posizione =", U);
}
```

Esempio

$$Costi = \begin{bmatrix} 1000 & 33 & 14 & 40 & 14 & 11 \\ 34 & 1000 & 21 & 13 & 20 & 23 \\ 14 & 21 & 1000 & 29 & 2 & 23 \\ 41 & 13 & 29 & 1000 & 27 & 30 \\ 15 & 20 & 2 & 27 & 1000 & 3 \\ 12 & 22 & 2 & 30 & 4 & 1000 \end{bmatrix}$$



Soluzione ottima
costo = 89

Per i più curiosi

https://www.youtube.com/watch?v=q8nQTNvCrjE&t=35s

Formulazione MZT

```
  Node  Left     Objective  IInf  Best Integer    Best Bound    ItCnt     Gap

     0     0       13.6000    19                   13.6000       35
     0     2       13.6000    19                   13.6000       35
                   0              0
Elapsed time = 0.21 sec. (0.67 ticks, tree = 0.02 MB, solutions = 0)
*   128    85      integral     0      104.0000       14.4000      316   86.15%
*   143    91      integral     0      103.0000       14.4000      464   86.02%
*   151    86      integral     0      101.0000       14.4000      396   85.74%
*   176    86      integral     0       80.0000       14.4000      537   82.00%
*   206    80      integral     0       76.0000       14.4000      627   81.05%
*   259    78      integral     0       75.0000       14.4000      706   80.80%
*   498    62      integral     0       39.0000       14.4000     1465   63.08%
```

Formulazione DFJ

```
        Nodes                                          Cuts/
   Node  Left     Objective  IInf  Best Integer    Best Bound    ItCnt     Gap

*     0+    0                          212.0000        0.0000           100.00%
*     0+    0                          174.0000        0.0000           100.00%
      0     0       39.0000    16      174.0000       39.0000       40   77.59%
*     0+    0                           46.0000       39.0000           15.22%
*     0+    0                           39.0000       39.0000            0.00%
      0     0        cutoff             39.0000       39.0000       40    0.00%
Elapsed time = 0.25 sec. (71.32 ticks, tree = 0.01 MB, solutions = 4)
```

Implementare in OPL la formulazione DFJ

Un modo per implementare in OPL la formulazione DFJ è quello di rappresentare i sottoinsiemi di V mediante i loro vettori caratteristici, ovvero passando come input al modello una matrice 0/1 W con tante colonne quanto sono i nodi del grafo e tante righe quanto sono i sottoinsiemi di V che bisogna considerare, con l'ovvio significato:

w_{ij} = 1 se il nodo j appartiene al sottoinsieme i

Tale matrice può essere calcolata offline e passata appunto come input insieme ad un vettore Card che ha tanti elementi quante sono le righe di W. L'elemento i di Card è il numero di «1» nella riga i di W e quindi rappresenta la cardinalità del sottoinsieme W_i
Ad esempio se |V|=n=4, W e Card assumono la forma

W
16
Sottoinsiemi

0	0	0	0	0
0	0	0	1	1
0	0	1	0	1
0	0	1	1	2
0	1	0	0	1
0	1	0	1	2
…	…	…		…
1	1	0	1	3
1	1	1	0	3
1	1	1	1	4

Card
16
Cardinalità dei sottoinsiemi

Avendo a disposizione W e Card , i vincoli di eliminazione dei sottocicli o i vincoli di taglio possono essere scritti facilmente

Esercizio per Casa……

Avendo a disposizione W e Card, i vincoli di eliminazione dei sottocicli o i vincoli di taglio possono essere scritti facilmente, scandendo opportunamente le righe di W

Vincoli di connessione nella formulazione DFJ

```
forall(w in Sottoinsiemi : Card[w] >=2 && Card[w]<=nNodi-2)
  sum(i in Nodi, j in Nodi : W[w][i]==1  &&  W[w][j]==0)X[i][j] <=Card[w]-1;
```

Vincoli di eliminazione sottocicli nella formulazione DFJ

```
forall(w in Sottoinsiemi : Card[w] >=2 && Card[w]<=nNodi-2)
  sum(i in Nodi, j in Nodi : W[w][i]==1  &&  W[w][j]==1  &&  i!= j  )X[i][j] <=Card[w]-1;
```

Esercizio per casa

9999	3	5	48	48	8	8	5	5	3	3	0	3	5	8	8	5
3	9999	3	48	48	8	8	5	5	0	0	3	0	3	8	8	5
5	3	9999	72	72	48	48	24	24	3	3	5	3	0	48	48	24
48	48	74	9999	0	6	6	12	12	48	48	48	48	74	6	6	12
48	48	74	0	9999	6	6	12	12	48	48	48	48	74	6	6	12
8	8	50	6	6	9999	0	8	8	8	8	8	8	50	0	0	8
8	8	50	6	6	0	9999	8	8	8	8	8	8	50	0	0	8
5	5	26	12	12	8	8	9999	0	5	5	5	5	26	8	8	0
5	5	26	12	12	8	8	0	9999	5	5	5	5	26	8	8	0
3	0	3	48	48	8	8	5	5	9999	0	3	0	3	8	8	5
3	0	3	48	48	8	8	5	5	0	9999	3	0	3	8	8	5
0	3	5	48	48	8	8	5	5	3	3	9999	3	5	8	8	5
3	0	3	48	48	8	8	5	5	0	0	3	9999	3	8	8	5
5	3	0	72	72	48	48	24	24	3	3	5	3	9999	48	48	24
8	8	50	6	6	0	0	8	8	8	8	8	8	50	9999	0	8
8	8	50	6	6	0	0	8	8	8	8	8	8	50	0	9999	8
5	5	26	12	12	8	8	0	0	5	5	5	5	26	8	8	9999

Matrice dei costi per un'istanza con 17 nodi (9999 sulla diagonale è un valore fittizio che indica l'assenza degli archi (i, i)).
Best Known Solution = 39