Ottimizzazione combinatoria

Branch and Bound per il TSP

Francesco Iodice

Anno 2020/2021

Sommario

- Travelling Salesman Problem
    - Branch and Bound
    - Rilassamento 1-albero
    - Branching sugli archi nel ciclo del 1-albero
    - Branching sul grado dei nodi
- Implementazione
    - Analisi risultati

Travelling Salesman Problem

Dato un grafo G(V, E) con costi c_{ij} associati agli archi, il Travelling salesman problem (TSP) richiede di determinare un insieme di archi $C^* \subset E$ che costituisca un ciclo hamiltoniano, ovvero un ciclo nel grafo in modo che ogni vertice $i \in V$ sia visitato esattamente una volta e il costo degli archi selezionati sia minimo.

Modello PLI

$$\min \sum_{i \in V} \sum_{j \in V} c_{ij} x_{ij}$$

$$\text{s.t.} \quad \sum_{j \in V} x_{ij} = 1 \qquad \forall j \in V \qquad (1)$$

$$\sum_{i \in V} x_{ij} = 1 \qquad \forall i \in V \qquad (2)$$

$$\sum_{i \in S} \sum_{j \in V \setminus S} x_{ij} \leq |S| - 1 \qquad \forall S \subset V, 2 \leq |S| \leq |V| - 1 \qquad (3)$$

$$x_{ij} \in \{0, 1\} \qquad \forall i, j \in V \qquad (4)$$

Travelling Salesman Problem

Un modo alternativo di proibire i sottocicli è quello di rimpiazzare i (3) con i seguenti vincoli:

$$\sum_{i \in S} \sum_{j \in V \setminus S} x_{ij} \geq 1 \qquad \forall S \subset V, 2 \leq |S| \leq |V| - 1 \qquad (5)$$

Questa famiglia di vincoli impone la connessione tra ogni sottoinsieme proprio di nodi S e l'insieme $V \setminus S$, **ovvero ogni sottoinsieme proprio di nodi** $S \subset V$ **deve essere connesso al resto del grafo**, impedendo quindi che il sottografo di archi $ij \in E : x_{ij} = 1$ sia costituito da componenti connesse separate.

Il numero dei vincoli per l'eliminazione delle componenti connessi separate, espressi tramite la famiglia di vincoli (3) o (5), **è esponenziale nelle dimensioni del problema**.

Il problema del commesso viaggiatore è *NP*-completo.

Travelling Salesman Problem

Nel caso particolare in cui il grafo $G(V, E)$ è non orientato il problema del commesso viaggiatore è detto **simmetrico**, il costo per andare in un nodo i ad un nodo j è uguale in entrambe le direzioni, $\forall \{i, j\} \in E, c_{ij} = c_{ji}$

Modello PLI

$$\min \sum_{ij \in E} c_{ij} x_{ij}$$

$$\text{s.t.} \quad \sum_{ij \in E} x_{ij} = 2 \qquad \forall i \in V \tag{6}$$

$$\sum_{i \in S} \sum_{j \in V \setminus S} x_{ij} \geq 2 \qquad \forall S \subset V, 2 \leq |S| \leq |V| - 1 \tag{7}$$

$$x_{ij} \in \{0, 1\} \qquad \forall \{i, j\} \in E \tag{8}$$

Branch and Bound

Gli algoritmi di tipo Branch and Bound si basano sui seguenti elementi:

- **bound**: determinare un Bound (o Upper Bound nel caso dei problemi di massimizzazione o Lower Bound nei problemi di minimizzazione), ovvero **la bontà della soluzione prodotta da un sottoproblema**, per **evitare** lo sviluppo completo di sottoalberi non promettenti.

- **branch**: costruzione dell'**albero di branch**, dopo la valutazione di un nodo, se la soluzione non è ammissibile per deve è possibile realizzare nuovi nodi figli con delle restrizioni ulteriori che sperabilmente rendano ammissibile una futura soluzione prodotta dai nodi discendenti.

- sapere identificare **quando la soluzione prodotta da un sottoproblema P_i è ammissibile per P.**

Branch and Bound

Algorithm 1 *BranchBoundTSP(P)*

Input: P instance of TSP problem
Output: z^* optimal objective function x^* optimal values of variables

```
1  P' ← Relaxation(P)
2  z* ← +∞, x* ← null, OpenNodes ← {P'}
3  x' ← Solve(P')
4  if x' is an hamiltonian cycle then
5      z* ← c(x) x* ← z'
6  while OpenNodes != ∅ do
7      P_i ← LowestLB(OpenNodes)
8      BranchingNodes ← Branch(P_i)
9      for every node x in BranchingNodes do
10         if x is an hamiltonian cycle and c(x) < z* then
11             z* ← c(x) x* ← x
12             remove from OpenNodes every P_i with LB(P_i) > z*
13         else if c(x) < z* then
14             OpenNodes ← OpenNodes ∪ {x}
15 return z*, x*
```

Schema generale raffinabile tramite le procedure:

- *Branch*(P_i)
- *Relaxation*(P)

Rilassamento 1-albero

Consideriamo una qualunque soluzione ammissibile per il TSP simmetrico: **il numero di lati incidenti su ciascun vertice è pari a 2** e, immaginando di rimuovere i due archi incidenti su un vertice a piacere (ad esempio il vertice x), quel che rimane è un albero ricoprente per il sottografo indotto da $V \backslash x$.

Pertanto, qualunque soluzione ha la seguente struttura:

1. l'insieme degli archi individuato crea un ciclo hamiltoniano;
2. ci sono due archi incidenti nel vertice x;
3. togliendo il vertice x rimane un albero ricoprente sul sottografo $V \backslash x$.

Il vincolo (1) è quello difficile da imporre, per cui lo si può rimuovere ed ottenere una soluzione rilassata nel seguente modo:

- calcolando il minimum spanning tree (MST) sul sottografo ottenuto eliminando il vertice x;
- aggiungendo i due archi di costo minimo incidenti in x. La soluzione così ottenuta è definita 1-albero

Rilassamento 1-albero

Definizione. *Un 1-albero di un grafo $G(V, E)$ è un suo sottografo connesso $T(V, E')$, con $E' \subset E$, tale che:*

- *un nodo $x \in V$ ha grado 2*
- *T contiene esattamente un ciclo che passa per x*

Avendo rimosso il vincolo (1) abbiamo ottenuto la possibilità di ottimizzare sugli 1-alberi anzichè sui cicli hamiltoniani. La famiglia degli 1-alberi contiene la famiglia dei cicli hamiltoniani come sottoinsieme stretto, infatti **ogni ciclo hamiltoniano su G è anche un suo 1-albero, ma non viceversa** quindi il problema di determinare l'1-albero di costo minimo di G è un rilassamento del TSP.

Rilassamento 1-albero

Di seguito è riportato un possibile modello PLI per l'1-albero dove $x = 1$ che mette in evidenza la relazione con il problema del TSP.

Modello PLI

$$\min \sum_{ij \in E} c_{ij} x_{ij}$$

$$\text{s.t.} \quad \sum_{1j \in E} x_{1j} = 2 \qquad \forall j \in V \qquad (9)$$

$$\sum_{i \in S} \sum_{j \in V \setminus S} x_{ij} \geq 1 \qquad \forall S \subset V, 2 \leq |S| \leq |V| - 1 \qquad (10)$$

$$x_{ij} \in \{0, 1\} \qquad \forall \{i, j\} \in E \qquad (11)$$

Proprietà. *Se T^* è un 1-albero di costo minimo di G, allora $T[V \setminus x]$ è un minimum spanning tree di $G[V \setminus x]$.*

Rilassamento 1-albero

Anche il problema del MST può essere espresso come modello di PLI

Modello PLI

$$\min \sum_{ij \in E} c_{ij} x_{ij}$$

$$\text{s.t.} \quad \sum_{ij \in E} x_{ij} = n - 1 \qquad \forall j \in V \qquad (12)$$

$$\sum_{i \in S} \sum_{j \in V \setminus S} x_{ij} \geq 1 \qquad \forall S \subset V, 2 \leq |S| \leq |V| - 1 \qquad (13)$$

$$x_{ij} \in \{0, 1\} \qquad \forall \{i, j\} \in E \qquad (14)$$

Il modello del MST mette in evidenza la relazione con il modello dell'1-albero, riportano la stessa funzione obbiettivo e una famiglia di vincoli analoghi.

Rilassamento 1-albero

Calcolo del lower bound per il problema originario Possiamo utilizzare la seguente procedura per realizzare un 1-albero ottimo $T*$

- **Passo 1.** Si risolva il problema MST sul grafo ottenuto scartando da $G = (V, A)$ il nodo x e tutti gli archi incidenti su di esso. Sia A_T la soluzione trovata;
- **Passo 2.** Si aggiungano ad A_T i due archi (x, k) e (x, h) a distanza minima tra tutti quelli incidenti sul nodo x .
- **Passo 3** Si restituisca $T^* = (V, E')$ con $E' = A_T \cup (x, k); (x, h)$.

In conclusione, l'1-albero di costo minimo fornisce un lower bound per il valore dell'ottimo del TSP simmetrico, il valore della soluzione trovata sarà sempre minore o uguale al valore della soluzione ottima del problema di partenza: $LB = c(T) \leq c(C^*)$.

Branching

Una soluzione ottima del rilassamento 1-albero non soddisfa, in generale, i vincoli di grado per qualche nodo, per cui un algoritmo branch-and-bound basato sul rilassamento 1-albero deve prevedere uno schema di branching che consenta di la generazione dei sottoproblemi P_i .

Supponiamo quindi di avere a disposizione un 1-albero ottimo T^* dobbiamo capire cosa rende non ammissibile per il problema P e **identificare uno schema di branch** che tenti di risolvere i difetti.

Branching sugli archi nel ciclo dell'1-albero

Dato che T^* è un 1-albero, per definizione contiene almeno un ciclo, se siamo fortunati tale ciclo è il ciclo hamilitoniano possiamo aggiornare il valore di z e chiudere il nodo, altrimenti possiamo quindi supporre che esista un nodo i in cui incidano più di due archi di T^* .

Siano $(a_1, a_2)(a_2, a_3) \ldots, (a_k, a_1)$ questi archi.

Lo schema prevede di costruire k (numero degli archi nel ciclo di T^*.) figli del nodo: in ciascuno dei figli si è fissato uno degli archi $(a_1, a_2)(a_2, a_3) \ldots, (a_k, a_1)$ come non appartenente al ciclo.

Si utilizzeranno gli E_0 e E_1 per fissare a priori alcuni archi a 1 oppure a 0.

Branching sugli archi nel ciclo dell'1-albero

Il primo nodo figlio viene ottenuto imponendo che in esso non sia presente l'arco (a_1, a_2) (cioè si impone $x_{a_1,a_2} = 0$), il secondo nodo figlio viene ottenuto imponendo che sia presente l'arco (a_1, a_2) ma che non sia presente l'arco (a_2, a_3) (cioè si impone $x_{a_1,a_2} = 0$,$x_{a_2,a_3} = 1$), e così via fino al k-esimo figlio in cui si impone che siano presenti tutti gli archi (a_j, a_{j+1}) con $j = 1,...,k-2$ ma non sia presente l'arco (a_k, a_1).

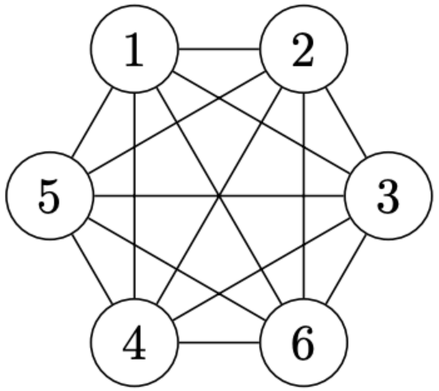
Il seguente ciclo $(a_1, a_2) \dots (a_4, a_1)$ genererebbe 4 problemi figli con i seguenti insiemi E_0 e E_1.

- Nodo 2: $E_0 = \{(a_1, a_2)\}, E_1 = \varnothing$
- Nodo 3: $E_0 = \{(a_2, a_3)\}, E_1 = \{(a_1, a_2)\}$
- Nodo 4: $E_0 = \{(a_3, a_4)\}, E_1 = \{(a_1, a_2), (a_2, a_3)\}$
- Nodo 5: $E_0 = \{(a_4, a_1)\}, E_1 = \{(a_1, a_2), (a_2, a_3), (a_3, a_4)\}$

In altre parole si fa in modo che in ciascuno dei figli sia assente almeno un arco del ciclo, il che esclude la presenza di tale ciclo in ciascuno dei nodi figli.

Branching sugli archi nel ciclo dell'1-albero

Esempio si consideri l'istanza di TSP simmetrico corrispondente ad un grafo completo $G(V, E)$



$$\begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \end{array} \begin{array}{c} \begin{array}{cccccc} 1 & 2 & 3 & 4 & 5 & 6 \end{array} \\ \begin{pmatrix} \infty & & & & & \\ 4 & \infty & & & & \\ 3 & 6 & \infty & & & \\ 7 & 9 & 8 & \infty & & \\ 5 & 1 & 4 & 2 & \infty & \\ 9 & 2 & 3 & 4 & 7 & \infty \end{pmatrix} \end{array}$$

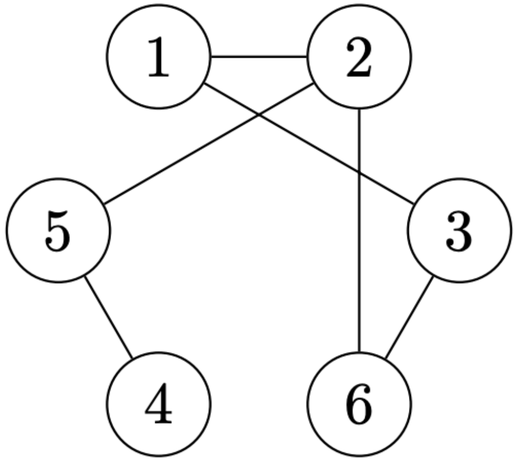
Utilizzando, arbitrariamente, $x = 1$, si determina l'**1-tree** di costo minimo formato dagli archi $T_1^* = \{(1,2), (1,3), (2,5), (2,6), (3,6), (4,5)\}$, dove $A_{T_1} = \{(2,5), (2,6), (3,6), (4,5)\}$ è l'albero di supporto a costo minimo determinato per il sottografo indotto dai nodi $V\backslash 1$, e (1, 2),(1, 3) sono i due archi di costo minimo incidenti sul nodo x = 1.

Branching sugli archi nel ciclo dell'1-albero

Si osserva che questo 1-albero non corrisponde ad una soluzione ammissibile per il TSP in quanto contiene il ciclo (1, 2, 6, 3, 1) - archi (1, 2),(2, 6),(3, 6),(1, 3).

-Si procedure quindi al branch, che dal nodo 1 produce i seguenti nodi figli, con i rispettivi insiemi E_0 e E_1.

- Nodo 2: $E_0 = \{(1,2)\}, E_1 = \varnothing$
- Nodo 3: $E_0 = \{(2,3)\}, E_1 = \{(1,2)\}$
- Nodo 4: $E_0 = \{(3,4)\}, E_1 = \{(1,2),(2,3)\}$
- Nodo 5: $E_0 = \{(4,1)\}, E_1 = \{(1,2),(2,3),(3,4)\}$



1-albero nodo 1

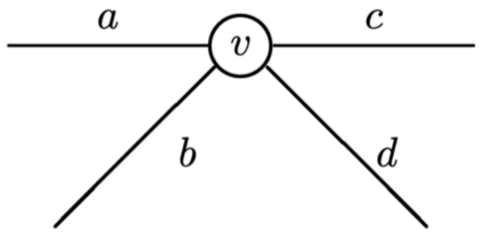
Risulta $LB_1 = \sum_{(i,j)\in T_1} c_{ij} = 15.$

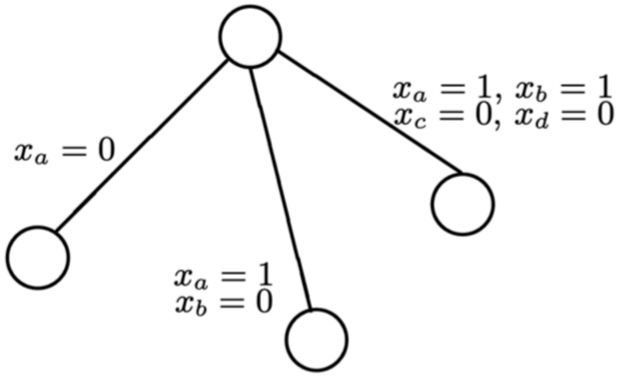
Branching sul grado dei nodi

Un altro fattore che possa rendere **non ammissibile** la nostra soluzione è il **grado dei nodi,** nella formulazione del problema dell'1-albero imponiamo che solo il nodo *x* deve avere grado 2.

Si sceglie un vertice $v \in T^*$ con grado ≥ 3 e due lati *a* e *b* incidenti su *v*; il problema è suddiviso in 3 sottoproblemi ottenuti vietando, a turno:

- il lato *a*
- il lato *b*
- tutti gli altri lati incidenti su *v* in modo da rispettare i vincoli di grado ogni nodo a 2.





Implementazione

Implementazione

Ogni sottoproblema, nonché nodo dell'albero di Branch, è rappresentato da un'istanza dell'oggetto **PNode**, con i seguenti campi.

```
class PNode:
    def __init__(self, V, E, E_0=[], E_1=[], node_x=1):
        self.P = None
        self.V = V
        self.E = E
        self.E_0 = E_0
        self.E_1 = E_1
        self.node_x = node_x
        self.LB = float('inf')
        self.solution = []
        self.name = ''
```

Implementazione - Rilassamento

PNode.P è un problema di programmazione lineare, rappresenta il modello dell'1-albero costruito in base agli insiemi E_0 e E_1.

```
def add_constraint_1_tree(self, x, proper_subset):
    self.P += lpSum(x[(self.node_x, j)] for j in self.V if self.node_x != j) == 2
    for S in proper_subset:
        V_S = list(set(self.V) - set(S))
        if 2 <= len(S) < len(self.V):
            vector = []
            for i in S:
                for j in V_S:
                    if i < j:
                        vector.append(x[(i, j)])
                    else:
                        vector.append(x[(j, i)])
            self.P += lpSum(vector) >= 1

    for edge in self.E_0:
        self.P += lpSum(x[(edge[0], edge[1])]) == 0
    for edge in self.E_1:
        self.P += lpSum(x[(edge[0], edge[1])]) == 1

    return self.P
```

Implementazione - Branching

Branching sugli archi nel ciclo del 1-albero

In base agli archi nel **ciclo,** vengono identificati gli insiemi E_0 e E_1 e costruiti i relativi sottoproblemi P_i.

P_i viene risolto e se è una soluzione ammissibile aggiungiamo P_i alla lista dei nodi aperti.

```
def branch(P, cycle):
    global nodes_generated
    subproblems = []
    for i, edge in enumerate(cycle):
        E_0, E_1 = [cycle[i]], cycle[:i]
        nodes_generated += 1
        P_i = PNode(copy.deepcopy(P.V), copy.deepcopy(P.E),
                    E_0=copy.deepcopy(P.E_0), E_1=copy.deepcopy(P.E_1),
                    name='P'+str(nodes_generated))
        P_i.update_E_set(E_0, E_1)
        P_i.build_LP_problem(proper_subsets_of_V)
        P_i.solve()
        if P_i.LB is not None:
            subproblems.append(P_i)
    return subproblems
```

Implementazione - Branch and Bound

L'algoritmo Branch and Bound per il TSP opera su una lista di **PNode**, i nodo aperti nell'albero di branch.

Si noti che l'input dell'algoritmo **P** è già una versione rilassata del problema.

```python
def branch_and_bound_tsp(P):
    PNodes, h_cycle, z = [P], [], float('inf')
    P.solve()

    if P.is_hamilton_cycle():
        h_cycle = P.solution, z = P.LB

    while len(PNodes) > 0:
        P_i = PNodes.pop(index_best_node(PNodes))
        branch_nodes = branch(P_i, find_cycle(P_i.solution))
        for p_node in branch_nodes:
            if p_node.is_hamilton_cycle():
                if p_node.LB < z:
                    z, h_cycle = p_node.LB, p_node.solution
                    closed_open_node(PNodes, z)
            elif p_node.LB < z:
                PNodes.append(p_node)
    return z, h_cycle
```

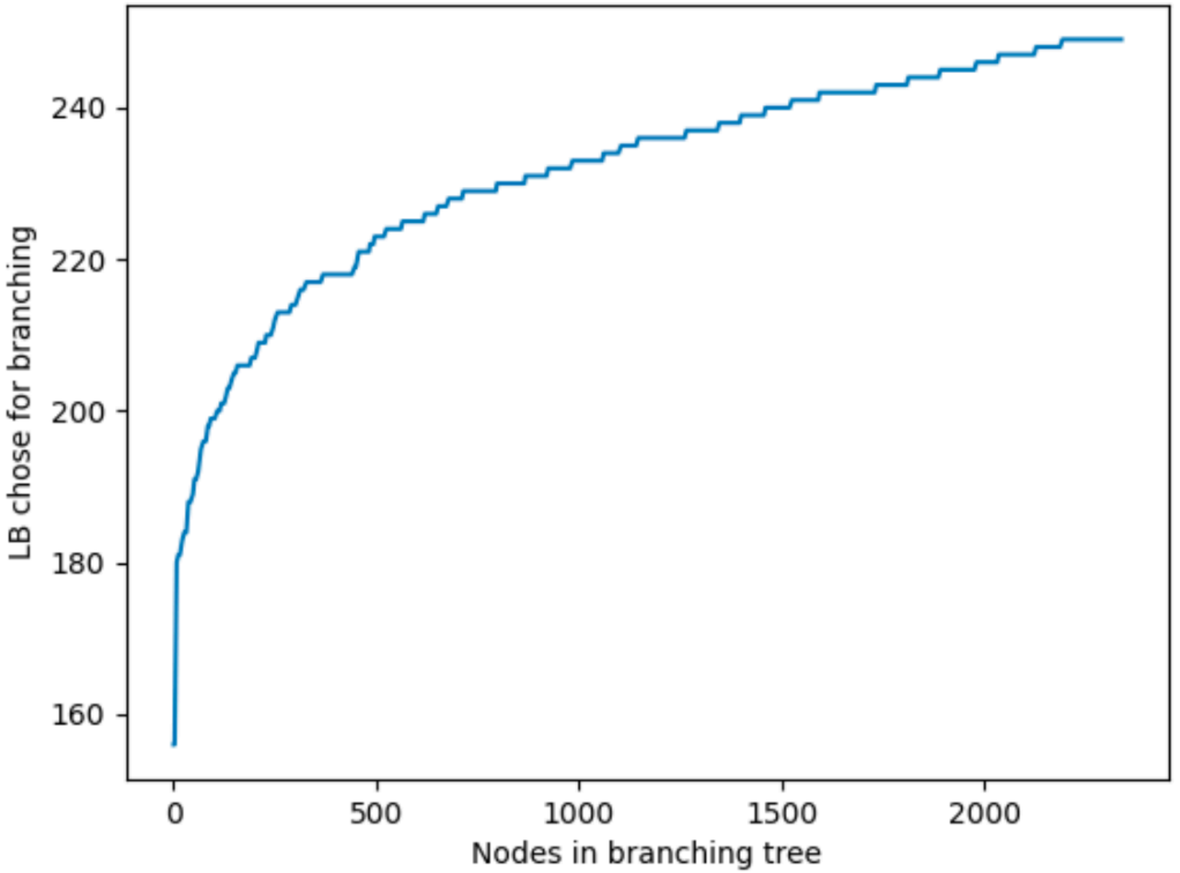
Risultati - Complessità

	Nodes generated in branching tree	Time (sec)
$\|V\| = 5$	7	0.09
$\|V\| = 6$	25	0.29
$\|V\| = 7$	172	1.84
$\|V\| = 8$	1169	18.75
$\|V\| = 9$	863	25.68
$\|V\| = 10$	2337	139.92
$\|V\| = 11$	8616	2480.89

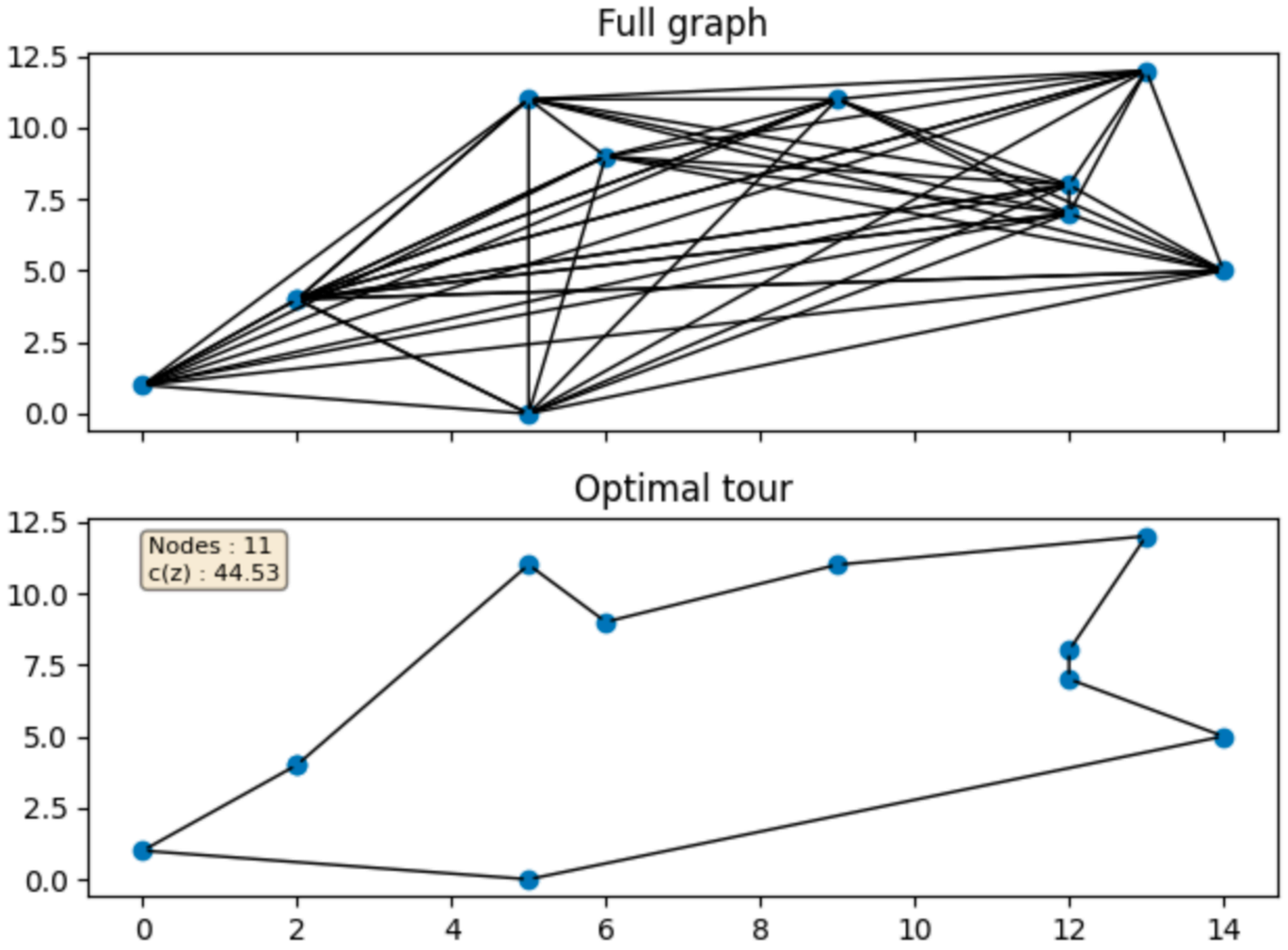
Risultati - Analisi

L'algoritmo tenta subito i percorsi più promettenti risolvendo i nodi con i Lower Bound inferiori, non trovando una soluzione ammissibile tenta altri percorsi relativi a nodi con valori di LB_i maggiori.

Sull'asse y è rappresentato il LB del nodo scelto, ad ogni iterazione, per fare branching.



Risultati - Interpretazione



Grazie per l'attenzione!