RELAZIONE PROGETTO AMOD ANNO 2019/2020

# Cutting Stock Monodimensionale

Autore:

**Alessio Mazzola, Matricola: 0279323**

# Indice

**1 Introduzione** **2**

**2 Specifiche del Problema** **3**
2.1 Problema generale: PLI e PL . . . . . 3
2.2 Problema Ristretto . . . . . 4

**3 Oracolo di generazione Colonne** **9**

**4 Round-UP e approssimazione** **12**

**5 Grafici e Commenti** **14**

# Capitolo 1

# Introduzione

Lo scopo del progetto è stato quello di andare a risolvere il problema del Cutting Stock Monodimensionale. La peculiarità di questo problema sta nel fatto che il numero di variabili del problema stesso può diventare intrattabile.

Nel problema del Cutting Stock monodimensionale, possiamo immaginare di dover acquistare dei *Paper Roll* i quali posseggono una lunghezza predeterminata $L$.

Una volta identificata la lunghezza, il Paper Roll stesso potrà essere tagliato in *Moduli* di lunghezze predefinite, i quali posseggono una certa domanda $D$, la quale dovrà necessariamente essere rispettata per il problema.

L'obiettivo del problema è quindi quello di andare a minimizzare il numero di *Paper Roll* utilizzati per andare a soddisfare la domanda degli utenti.

L'idea utilizzata per la risoluzione del problema è quella dell'utilizzo dei *Cutting Pattern*, anche chiamati *Modalità di Taglio*, i quali rappresentano dei diversi possibili modi per andare a tagliare il Paper Roll al fine di ottenere i moduli richiesti per soddisfarne la domanda.

Una modalità di Taglio risulta essere ammissibile quando la somma delle lunghezze di tutti i moduli che compongono la modalità di taglio stessa sia minore o uguale alla lunghezza del Paper Roll. In particolare, se la somma è minore, allora il Cutting Pattern andrà a produrre uno *scarto* (anche chiamato *sfrido*), altrimenti non vi sarà alcuno scarto prodotto. In generale è possibile scrivere qualunque tipo di cutting pattern ammissibile come una colonna $A_j$ che presenta componenti intere, e con un numero di componenti pari al numero dei moduli.

$$A_j = \begin{bmatrix} a_{1j} \\ a_{2j} \\ \dots \\ a_{mj} \end{bmatrix}$$

# Capitolo 2

# Specifiche del Problema

In questo capitolo andiamo a discutere delle specifiche del problema del Cutting Stock, andando prima ad introdurre il problema generale e successivamente introducendo il problema ridotto, che appunto permette la risoluzione di questa tipologia di problemi, andando a rendere il numero di variabili da considerare **trattabile**.

## 2.1 Problema generale: PLI e PL

Il problema associato al Cutting Stock, in generale, sarà un problema di programmazione lineare **intera** poiché guardando il problema da uno scenario realistico, è normale aspettarsi che il numero di *Paper Roll* che si necessitano per soddisfare la domanda debba essere intero.

Un aspetto importante nella definizione del problema risulta essere la definizione delle variabili che lo compongono, in particolare quindi avremo:

$$x_j \in \mathbb{Z}_+; j = 1, ..., |T|$$

La variabile $x_j$ rappresenta il numero di Paper roll tagliati attraverso la modalità di taglio j.

Non è difficile comprendere che, per un problema generico, l'insieme T può avere una cardinalità elevatissima e quindi, proprio per questo motivo, il problema risulta avere un numero di variabili intrattabile. In particolare le possibili modalità di taglio sono individuate attraverso la combinazione degli elementi, quindi si ha:

$$|T| = \binom{n}{k} = \frac{n!}{k!(n-k)!}$$

Come detto in precedenza, l'obiettivo del problema è quello di andare a minimizzare il numero di Paper Roll utilizzati per soddisfare la doman-

da dei diversi moduli. A questo scopo la funzione obiettivo del problema risulta essere definita nel seguente modo:

$$z = \min \sum_{j \in T} x_j$$

Per garantire invece che la domanda sia soddisfatta, i vincoli del problema saranno individuati dalle seguenti disequazioni:

$$\sum_{j \in T} A_j x_j \geq d = \begin{pmatrix} d_1 \\ d_2 \\ ... \\ d_m \end{pmatrix}$$

Possiamo quindi a questo punto ricavare la formulazione *PLI* del problema:

$$z = \min(\sum_{j \in T} x_j : \sum_{j \in T} A_j x_j \geq d, x_j \in \mathbb{Z}_+, j = 1, ..., |T|)$$

Se le componenti del vettore $d$ assumono valori grandi rispetto al numero di moduli che ciascuna modalità di taglio può produrre, possiamo allora aspettarci che i valori assunti da $x_j$ siano dei valori interi decisamente grandi.

Per semplicità possiamo quindi andare a considerare il *Rilassamento Lineare* del problema, individuato dalla seguente formulazione:

$$z^c = \min(\sum_{j \in T} x_j : \sum_{j \in T} A_j x_j \geq d, x_j \in \mathbb{R}_+, j = 1, ..., |T|)$$

## 2.2 Problema Ristretto

Andando a prendere in considerazione il problema *Ristretto*, questo permette di considerare un insieme ristretto delle possibili modalità di taglio. In particolare questo risulta essere fondamentale nella risoluzione del problema in quanto permette di lavorare con un problema che ha un numero di variabili *trattabile* rendendo possibile la risoluzione del problema.

L'approccio utilizzato all'interno del progetto va a considerare come pattern di taglio iniziale, il pattern ricavato andando a considerare i moduli singolarmente, in modo tale da andare a ricavare una matrice che avrà tutti gli elementi nella diagonale:

$$F = \begin{bmatrix} f_{11} & 0 & \dots & 0 \\ 0 & f_{22} & \dots & 0 \\ 0 & 0 & \ddots & 0 \\ 0 & 0 & 0 & f_{nm} \end{bmatrix}$$

Per la generazione della matrice iniziale dei pattern di taglio è stato sviluppato il metodo *determineInitialPattern(...)* che riportiamo di seguito:

```
def determineInitialPattern(L, listOfModule):
    ##Questo metodo serve per inizializzare la matrice B
       delle modalita'
    ##di taglio possibili data la lunghezza L del paper
       Roll e
    ##la lista di tutti i moduli.

    ##Inizialmente andiamo a considerare solamente tutt

    #L: int --> Lunghezza del paperRoll
    #listOfModule: list --> [Lunghezze di tutti i moduli
       del problema]
    #return B --> matrince nxn di tutti i moduli di taglio

    n = len(listOfModule)
    initialLenght = L

    ##Liste di appoggio per il metodo.
    B = []
    A = []

    counter = 0

    for y in range(0, n):
        while L - listOfModule[y] >= 0:
            L = L - listOfModule[y]
            counter += 1
        for x in range (0, n):
            if x == y:
                A.append(counter)
            else:
                A.append(0)
        B.append(A)
        A = []
        counter = 0
        L = initialLenght

    return B
```

Il problema ristretto va quindi a rappresentare un sottoproblema in cui abbiamo una funzione obiettivo identica a quella del problema generale, ma anziché andare a considerare tutte le modalità di taglio possibili all'interno dell'insieme T, ne consideriamo solamente alcune che saranno racchiuse all'interno dell'insieme F, e quindi avremo:

$$F \subseteq T$$

In questo caso, il vincolo di domanda del problema ristretto sarà individuato da:

$$\sum_{j \in F} A_j x_j \geq d$$

Possiamo a questo punto andare a ricavare la formulazione lineare del problema ristretto, identificata da:

$$z^{ristretto} = \min(\sum_{j \in F} x_j : \sum_{j \in F} A_j x_j \geq d, x_j \in \mathbb{R}_+, j = 1, ..., |F|)$$

Per quanto riguarda la soluzione del problema ristretto, è stato invece sviluppato il metodo *resolvePrimal(...)* che riportiamo di seguito:

```
1  def resolve_primal(listOfDemand, cutScheme):
2      """Questo metodo va a risolvere il problema primale. In
           particolare da questo
3      andiamo a ritornare i seguenti elementi:
4      Lp_prob : Permette di ricavare la soluzione ottima del
          problema
5      B : Lista di elementi utilizzata come funzione
          obiettivo nel problema duale
6      C : Vettore dei costi associati al problema.
7      redueced_cost: Identifica il vettore dei costi ridotti
          del problema primale.
8      """
9      ## Liste di appoggio per il metodo.
10     B = []
11     C = []
12     reduced_cost = []
13     isOpt = 0
14     #Creazione del problema di programmazione lineare
          intera
15     Lp_prob = p.LpProblem('Primal_Problem', p.LpMinimize)
16     ##Creazione delle variabili
17     xs = [p.LpVariable("x{}".format(i), lowBound = 0, cat='
          Continuous') for i in range(len(cutScheme))]
18     ##Funzione obiettivo:
19     total_prof = sum(x for x in xs)
20     Lp_prob += total_prof
21     ##Diseguaglianze del problema:
22     counter = 0
23     for x in range(len(cutScheme[0])):
24         Lp_prob += sum (h * cut[x] for h ,cut in zip(xs,
              cutScheme)) >= listOfDemand[x] ##Questo funziona
               per il metodo add
25         counter += 1
26     #Solver
27     status = Lp_prob.solve()
```

```
28      print(p.LpStatus[status])
29      print("Objective value:", p.value(Lp_prob.objective))
30      print ('\nThe values of the variables : \n')
31      for v in Lp_prob.variables():
32          reduced_cost.append(v.dj)
33          C.append(v.varValue)
34          print(v.name, "=", v.varValue)
35      for name, c in list(Lp_prob.constraints.items()):
36          B.append(c.pi)
37
38      ##controllo se la soluzione del primale e' ottima
           tramite il vettore dei costi ridotti.
39      print('\nVettore dei costi ridotti: \n')
40      print(reduced_cost)
41      print('\n')
42      if(min(reduced_cost) >= 0):
43          isOpt = 1
44          return Lp_prob, B , C , isOpt
45      return Lp_prob, B , C , isOpt
```

**OSSERVAZIONE:** La soluzione che viene ottenuta andando a considerare il problema ristretto, e quindi considerando il sottoinsieme F, risulta essere comunque una soluzione ammissibile anche per il problema generale. Possiamo infatti vedere la soluzione del problema non ristretto come una soluzione del seguente tipo:

$$\bar{x} = (x_F : 0)$$

dove $x_F$ identifica le soluzioni del problema ristretto mentre andiamo ad imporre a 0 le soluzioni del problema generale che non sono state considerate all'interno di F.

Andando a considerare il problema duale associato al problema ristretto, il quale avrà la seguente formulazione:

$$\max(y^T d : y^T A^F \leq C^F, y \geq 0^m)$$

All'interno di questo problema il numero di vincoli sarà determinato dal numero di moduli richiesti dal problema generale. $C^F$ rappresenta un vettore formato da soli 1 con grandezza pari alla cardinalità dell'insieme F. Possiamo inoltre scrivere il vincolo j-esimo nel seguente modo:

$$y^T A_j \leq 1$$

Tornando a considerare il problema ristretto, siamo invece interessati a capire come possiamo esprimere il vettore dei *Costi Ridotti* per la generica variabilie $x_j$. Possiamo quindi scrivere:

$$\delta_y = c_j - y^{*T} A_j = 1 - \sum_{i=1}^{m} a_{ij} y_i^*$$

Questa espressione risulta ovviamente collegata ai vincoli del problema duale, infatti possiamo scrivere:

$$y^T A_j \leq 1 \iff (1 - \sum_{i=1}^{m} a_{ij} y_i) \geq 0$$

Ci stiamo quindi chiedendo quale condizione ci assicura che la soluzione del problema ristretto sia una soluzione ottima del problema continuo nella sua forma generale.

Notiamo che:

$$\nexists j \in T - F : \delta_j < 0$$

In particolare se $\delta_j < 0$ allora $\bar{x}$ non potrà essere ottima per il problema generale. Stiamo quindi dicendo che non esiste nessuna modalità di taglio tra quelle che abbiamo generato, e quindi tra quelle che appartengono solamente all'insieme T ma che non appartengono all'insieme F, che produce un beneficio per il problema ristretto.

Guardando in modo equivalente il problema duale stiamo dicendo che siamo interessati ad analizzare un numero di vincoli pari al numero delle modalità di taglio del problema ristretto, quindi equivalentemente per il duale possiamo chiederci se:

$\nexists j \in T - F : \delta_j < 0$ : il vincolo j risulta essere violato da $y^*$ del problema ristretto

Se questa non esiste, allora la soluzione che stiamo considerando è una soluzione ottima del problema duale generale. Supponendo quindi che $y^*$ sia una soluzione ottima del problema duale nella versione ristretta. Questo significa che tutti i vincoli del duale, cioè:

$$y^{*T} A_j \leq 1 \forall j \in T$$

sotto questa condizione, la soluzione $\bar{x}$ del problema ristretto risulta essere ottima anche per il problema di di Cutting Stock rilassato. Se invece $y^*$ dovesse violare qualche vincolo, allora questo significherebbe che:

$$\exists h \in T - F : y^{*T} A_h > 1$$

Quindi esiste una variabile, tra quelle non generate, che ha un costo ridotto negativo. Allora, in questo caso, $\bar{x}$ non può essere una soluzione ottima per il problema generale di CSP rilassato e quindi è conveniente introdurre la modalità di taglio $h$ all'interno del problema ristretto, in modo da identificare una nuova soluzione.

# Capitolo 3

# Oracolo di generazione Colonne

In questo capitolo andiamo ad introdurre l'*Oracolo di generazione Colonne*, il quale avrà un ruolo fondamentale nella risoluzione del problema poiché permetterà di andare a determinare nuovi pattern di taglio che possono essere aggiunti al problema ristretto in modo tale da ottenere, dopo diverse iterazioni dell'algoritmo, la soluzione del problema.

L'oracolo di generazione colonne prende anche il nome di *Pricing Problem* ed il suo scopo è quello di determinare una variabile con un costo ridotto strettamente negativo.

La prima cosa da comprendere è come viene identificata una modalità di taglio ammissibile, difatti deve essere rispettato il seguente vincolo:

$$\sum_{i=1}^{m} \alpha_i l_i \geq L$$
$$\alpha \in \mathbb{Z}_+^m$$

Nella formula $\alpha$ rappresenta la modalità di taglio, cioè l'incognita che siamo intenzionati a ricavare, mentre $l_i$ rappresenta la lunghezza del modulo i-esimo ed L invece rappresenta la lunghezza del *Paper Roll*. In particolare, quindi, una modalità di taglio risulterà essere ammissibile solamente se il vincolo risulta essere rispettato. Siamo quindi fondamentalmente interessati al valore dell'espressione:

$$y^{*T} A_h$$

Possiamo a questo punto andare a definire il problema di pricing nel seguente modo:

$$w = \max(y^{*T}\alpha : \sum_{i=1}^{m} \alpha_i l_i \geq L, \alpha \in \mathbb{Z}_+^m)$$

Notiamo inoltre come il problema di pricing sia effettivamente un problema di Knapsack Intero. Gli scenari che possono accadere sono quindi i seguenti:

- w $\leq 1$ : questo significa che non vi sono altre modalità di taglio che possono migliorare la soluzione ottenuta nel problema ridotto. In altre parole si ha:
$$y^{*T} A_h \leq 1$$
- w > 1 : questo significa che la soluzione del problema ridotto non era una soluzione ottima e che quindi possiamo aggiungere la modalità di taglio al problema e ripetere i passaggi effettuati per ricavare una nuova soluzione. In particolare, quindi:
$$y^{*T} A_h > 1$$

All'interno del progetto, l'oracolo di generazione colonne è stato sviluppato attraverso il metodo *resolvePricing(...)* che riportiamo di seguito:

```
1  def resolve_pricing(L, listObjectiveFunction, listInequity)
       :
2  ##Questo metodo va a risolvere il sottoproblema di knapsack
3      ##derivante dal CSP rilassato, andando a restituire la
           modalita'
4      ##di taglio che deve entrare in base.
5      ##Un pattern entra in base solamente se il valore della
           sua
6      ##funzione obiettivo e' > 1.
7      #L --> int: Lunghezza del Roll
8      #listObjectiveFunction: --> list: [valori della
           funzione obiettivo] --> i valori sono individuati
           dai valori del problema duale
9      #listInequity --> list: [valori della diseguaglianza]
10     ##return A --> list: [Nuova Cutting pattern]
11
12     ##Creazione della lista per la nuova base.
13     A = []
14
15     #Creazione del problema di programmazione lineare
           intera
16     Lp_prob = p.LpProblem('Pricing Problem', p.LpMaximize)
17
18     ##Creazione delle variabili
19     xs = [p.LpVariable("x{}".format(i), lowBound = 0, cat='
           Integer') for i in range(len(listObjectiveFunction))
           ]
20
21     ##Funzione obiettivo:
```

```
22      total_prof = sum(x * obj for x,obj in zip(xs,
            listObjectiveFunction))
23      Lp_prob += total_prof
24
25      ##Diseguaglianze del problema:
26      total_weight = sum(x * w for x,w in zip(xs,
            listInequity))
27      Lp_prob += total_weight <= L
28
29      #Solver
30      status = Lp_prob.solve()
31      print(p.LpStatus[status])
32      print("Objective value:", p.value(Lp_prob.objective))
33      print ('\nThe values of the variables : \n')
34      for v in Lp_prob.variables():
35          print(v.name, "=", v.varValue)
36
37      ##Verifico se il valore della funzione obiettivo e'
            maggiore di 1,
38      ##In questo caso la lista A diventera' la nuova base da
            aggiungere
39      ##Al problema.
40      if p.value(Lp_prob.objective) > 1:
41          for v in Lp_prob.variables():
42              A.append(int(v.varValue))
43
44      return A, p.value(Lp_prob.objective)
```

# Capitolo 4

# Round-UP e approssimazione

In questo capitolo andiamo andiamo a parlare del round-up della soluzione del problema del Cutting Stock e della soluzione approssimata ottenuta attraverso questo metodo.

Possiamo dire che la soluzione del problema di Cutting Stock rilassato è in generale una soluzione frazionaria. Per questo motivo siamo quindi interessati ad introdurre una approssimazione al fine di rendere la soluzione intera.

In particolare, poiché il vincolo di domanda deve essere comunque soddisfatto, l'unico arrotondamento possibile risulta essere quello per eccesso, infatti considerando un arrotondamento per difetto, il vincolo di domanda risulterebbe non più rispettato; inoltre l'arrotondamento per eccesso risulta inoltre essere ovviamente una soluzione ammissibile per il problema di Cutting Stock non rilassato (quindi per la sua formulazione PLI).

Volendo comprendere il grado di errore che si commette andando ad eseguire l'approssimazione, possiamo scrivere:

$$z^c \leq \lceil z^c \rceil \leq z$$

essendo infatti $\lceil \bar{x} \rceil$ una soluzione ammissibile per il CSP, allora questo deve valere anche per:

$$z \leq e^T \lceil \bar{x} \rceil = \sum_{j \in \beta} \lfloor \bar{x}_j \rfloor + \phi_j = \sum_{j \in \beta} \lfloor \bar{x}_j \rfloor + \sum_{j \in \beta} \phi_j$$

In particolare $\phi_j$ rappresenta la parte frazionaria della componente j-esima mentre $\beta$ rappresenta la base ottima. Notiamo inoltre che si ha:

$$\sum_{j \in \beta} \lfloor \bar{x}_j \rfloor \leq z^c \leq z; \sum_{j \in \beta} \phi_j < m$$

Date queste considerazioni, possiamo dunque scrivere:

$$\sum_{j\in\beta}\lfloor\bar{x}_j\rfloor + \sum_{j\in\beta}\phi_j \leq z + (m-1)$$

e quindi in conclusione possiamo affermare che la soluzione $\lceil\bar{x}\rceil$ sarà compresa tra:

$$z \leq e^T\lceil\bar{x}\rceil \leq z + (m-1)$$

Andando quindi a definire l'*Errore Assoluto*, questo sarà indivituato da:

$$ERR_{abs} = z^{approssimato} - z$$

Quindi l'errore assoluto sarà sicuramente $\leq (m-1)$. Per quanto riguarda l'*Errore Relativo*, questo risulta essere definito nel seguente modo:

$$ERR_{rel} = \frac{z^{approssimato} - z}{z^{approssimato}}$$

ed in particolare, possiamo notare come questo errore abbia un valore compreso tra 0 ed 1. Eseguendo alcuni calcoli possiamo notare che:

$$\frac{z^{approssimato} - z}{z^{approssimato}} \leq \frac{m-1}{z^{approssimato}} \leq \frac{m-1}{\lceil z^c\rceil}$$

Quindi l'errore relativo che commettiamo dipenderà essenzialmente da quanto grande è $\lceil z^c\rceil$ rispetto a (m-1).

L'errore può essere più o meno accettabile, questa considerazione dipende ovviamente il grado di accettazione per l'istanza del problema presa in considerazione.

All'interno del progetto, l'approssimazione è stata eseguita attraverso il metodo *roundUpSolution(solutionList)* che riportiamo di seguito:

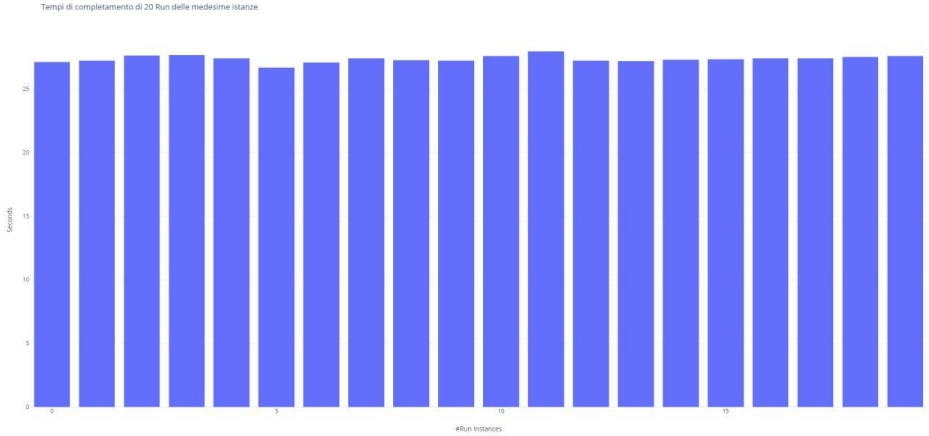
```
1  def roundUpSolution(solutionList):
2      ##Questo metodo si occupa di eseguire il rounding
3      ##up della solzione se questa non dovesse essere
4      ##intera.
5
6      ##solutionList: list[] --> lista di valori della
           soluzione (interi e non)
7      ##return: IntSolution --> valore della solzione intera.
8      solValue = 0
9      for x in solutionList:
10         solValue += math.ceil(x)
11
12     return solValue
```

Per quanto riguarda invece la soluzione ottima z, questa viene direttamente calcolata dal solover *Pulp* utilizzato all'interno del progetto.

# Capitolo 5

# Grafici e Commenti

In questa sezione andiamo a presentare alcuni grafici ed a commentare l'esecuzione dell'algoritmo sia in termini temporali che per quanto riguarda l'errore Assoluto e Relativo delle diverse istanze di prova.
Per quanto riguarda i tempi di esecuzione, è stato anzitutto testato se l'algoritmo avesse un tempo di completamento stabile nel tempo. A questo scopo è stata eseguita la run di tutte le istenze 20 volte consecutive. I risultati ottenuti sono visibili nel grafico in figura:




Come possiamo notare, i tempi di completamento per tutte le run presentano gli stessi valori con un minimo margine d'errore.
Per quanto riguarda invece l'errore medio e assoluto, questo è stato calcolato solamente su istanze tali per cui fosse stato possibile ricavare la soluzione ottima. In particolare, queste istanze sono individuate da moduli richiesti che risultano essere divisori della lunghezza totale del Paper Roll. Riportiamo ora in figura i grafici per alcune istanze con i valori calcolati per l'*Errore Assoluto* e l'*Errore Relativo*:

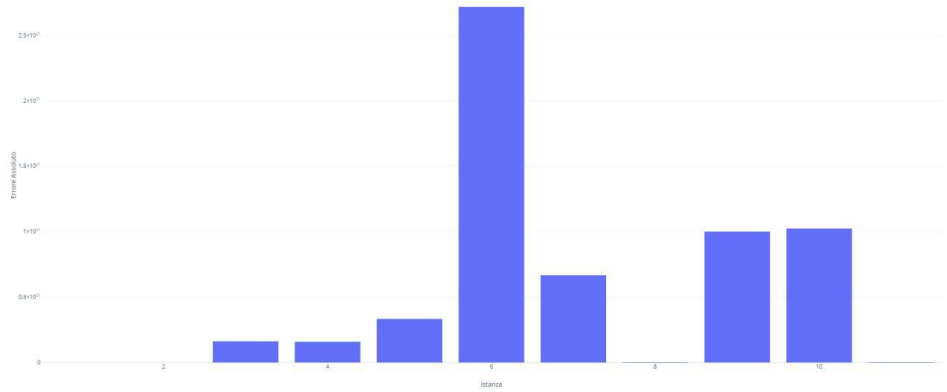



Figura 5.1: Errore Assouluto calcolato con le istanze di Test

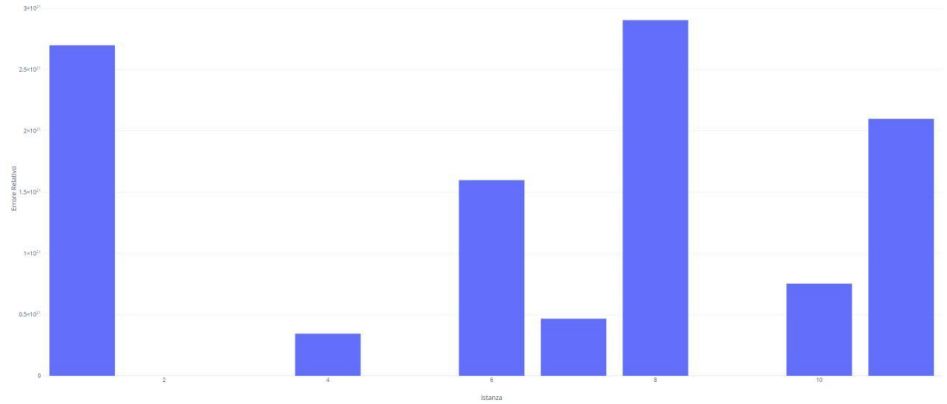



Figura 5.2: Errore Relativo calcolato con le istenze di Test

Come possiamo vedere, per le istanze selezionate, entrambi i valori dell'errore Assoluto e Relativo risultano essere accettabili.