Implementazione Algoritmo del Sottogradiente

Filippo Vajana

AA 2017/2018

Sommario

Il lavoro consiste nell'implementazione dell'algoritmo del sottogradiente per la risoluzione del problema Lagrangiano duale associato al rilassamento del modello intero per il problema *Uncapacitated Facility Location.*
Il modell primale del problema *UFL* e il vincolo da rilassare sono stati scelti prendendo a riferimento quanto fatto da Laurence A. Wolsey all'interno del libro *"Integer Programming"* [1].

1 Introduzione

Un problema economico di grande impatto pratico nella gestione di una attività commerciale riguarda il posizionamento delle *facility* (impianti produttivi, magazzini, etc.) per soddisfare la domanda minimizzando i costi, rappresentati dai costi *fissi* per l'apertura di una facility e dai costi *di trasporto* per la fornitura del servizio ai clienti.
Quando gli impianti in esame hanno una capacità produttiva limitata il problema prende il nome di *Capacitated Facility Location Problem.*

2 Problema Primale

$$
\begin{aligned}
\text{z} = \max \quad & \sum_{i \in M} \sum_{j \in N} c_{ij} x_{ij} - \sum_{j \in N} f_j y_j & & \\
\text{s.t.} \quad & \sum_{j \in N} x_{ij} = 1 & & \text{for } i \in M \\
& x_{ij} - y_j \leq 0 & & \text{for } i \in M,\, j \in N \\
& x \in \mathbb{R}^{|M| \times |N|}, y \in \mathbb{B}^{|N|} & &
\end{aligned}
\tag{1}
$$

dove:

- **M**: insieme dei clienti
- **N**: insieme delle località in esame
- $\mathbf{c_{ij}}$: profitto nel servire il cliente i dalla località j
- $\mathbf{f_j}$: costo nell'aprire una facility nella località j
- $\mathbf{x_{ij}}$: frazione della domanda del cliente i soddisfatta dalla località j
- $\mathbf{y_j}$: indica se nella località j è aperta una facility.

Nel modello illustrato sopra il primo vincolo assicura che la domanda di ogni cliente venga soddisfatta interamente.
Il secondo vincolo garantisce invece che nessun cliente possa venir servito da una località nella quale non è stata aperta una facility.

3 Problema Rilassato

$$\begin{aligned} \text{z(u)} = \max \quad & \sum_{i\in M}\sum_{j\in N} c_{ij}x_{ij} - \sum_{j\in N} f_j y_j + \sum_{i\in M}\sum_{j\in N} u_i(1-x_{ij}) \\ \text{s.t.} \quad & x_{ij} - y_j \leq 0 \qquad\qquad \text{for } i \in M,\, j \in N \\ & x \in \mathbb{R}^{|M|\times|N|}, y \in \mathbb{B}^{|N|} \end{aligned} \tag{2}$$

Il rilassamento lagrangiano è stato ottenuto *"portando"* il vincolo di soddisfacimento della domanda $\sum_{j\in N} x_{ij} = 1$ all'interno della funzione obiettivo con un fattore $\mathbf{u_i}$ di penalizzazione.
L'operazione comporta una semplificazione del modello al costo di un peggioramento della soluzione ottima ottenibile dal modello rilassato.

4 Problema Lagrangiano Duale

$$w_{LD} = \min_{u\geq 0} \quad z(u) \tag{3}$$

La soluzione del rilassamento lagrangiano di cui sopra, fornisce un *upper bound* al valore della soluzione ottima del problema originale. Per trovare il vettore $\mathbf{u}$ dei *moltiplicatori di Lagrange* che determina la migliore approssimazione è necessario risolvere il problema *lagrangiano duale* $\mathbf{w_{LD}}$.

5 Algoritmo del Sottogradiente

Di seguito è mostrato lo pseudocodice per l'algoritmo implementato.

Algorithm 1 Algoritmo del Sottogradiente

Input: $z(u)$ {funzione da massimizzare}
Input: u^0 {punto iniziale}
Input: T {limite iterazioni senza miglioramento}
Input: α_0 {parametro d'arresto}

$t \leftarrow 1$
$\alpha_t \leftarrow 2$
$z^\star \leftarrow \infty$
while $\alpha_t > \alpha_0$ **do**
  dato u^{t-1}, calcola la soluzione ottima x^t del problema Lagrangiano

$$z(u) = \max_x c^T x + u^T(d - Dx)$$

  dato il punto x^t, calcola sottogradiente s^t di $z(u)$

$$s^t \leftarrow d - Dx^t$$

  if $s^t == 0$ **then**
    return $z^\star$
  end if
  calcola la dimensione del passo di ricerca

$$\theta^t \leftarrow \alpha_t \frac{|\bar{z} - z(u)|}{||s^t||^2}$$

  con $\bar{z}$ miglior soluzione ammissibile nota del problema originale
  aggiorna il vettore dei moltiplicatori

$$u^t \leftarrow \max\{u^{t-1} - \theta^t s^t, 0\}$$

  if $z(u) < z^\star$ **then**
    $z^\star \leftarrow z(u)$ {aggiornamento upper-bound}
    $t \leftarrow 0$
  end if
  if $t = T$ **then**
    $\alpha_t \leftarrow \alpha_t/2$ {riduce moltiplicatore passo ricerca}
    $t \leftarrow 0$
  end if
  $t \leftarrow t + 1$
end while
return $z^\star$

6 Sperimentazione

6.1 Metodologia di prova

Nelle successive sezioni verranno illustrati tutti i dettagli riguardanti le procedure e gli accorgimenti usati per la raccolta dei dati sperimentali.

Tecnologie usate Per la realizzazione del progetto si è fatto affidamento su tecnologie consolidate, liberamente accessibili e con una buona documentazione a supporto.
La principale è rappresentata dalla piattaforma **Anaconda** (*v4.5.11*), la quale mette a disposizione una vasta raccolta di strumenti software utili per applicazioni nei campi della *data science*, del *machine learning* e del calcolo scientifico in generale.
Per sfruttare al massimo le potenzialità della piattaforma tutto il codice sorgente è stato scritto in **Python** (*v3.6.6*), avvalendosi dei seguenti *package*:

- **PuLP** (*v1.6.8*): scrittura dei modelli e successiva codifica in linguaggio compatibile con il solutore **COIN-OR** (*v1.16*)
- **Pandas** (*v0.23.4*): usato per organizzare in forma strutturata e analizzare i risultati sperimentali
- **NumPy** (*v1.15.2*): contiene funzioni indispensabili per il calcolo scientifico in linguaggio Python
- **Matplotlib** (*v2.2.2*): usato per la rappresentazione grafica dei risultati sperimentali.

La parte di organizzazione e rappresentazione grafica dei risultati è stata realizzata sotto forma di *notebook* **Jupyter** (*v5.6*) per consentire una migliore interazione.

Organizzazione progetto Il codice sorgente del progetto è stato diviso in moduli funzionali:

- **benchmark.py**: classi e funzioni necessarie ad eseguire una sessione di prove sperimentali.
- **data_utils.py**: funzioni utili alla generazione dei dati propri di una istanza di test
- **model.py**: in questo modulo è presente il codice che implementa i modelli dei problemi
- **solver_utils.py**: funzioni *helper* per l'inizializzazione dei modelli e la loro risoluzione
- **subgradient.py**: implementazione completa dell'algoritmo di sottogradiente.

Istanza di test Parlando di *istanza di test* si fa riferimento ad una specifica combinazione dei parametri base necessari per inizializzare e risolvere una istanza del problema in esame. Una istanza di test è descritta dalla tupla

$$\langle id, c_{num}, l_{num}, U, C, F \rangle \tag{4}$$

dove i termini hanno il seguente significato:

- **id**: identificativo unico per l'istanza di test
- $\mathbf{c_{num}}$: numero di clienti a cui erogare il servizio
- $\mathbf{l_{num}}$: numero di locations da prendere in considerazione
- **U**: vettore dei moltiplicatori di Lagrange
- **C**: matrice dei profitti *cliente - location*
- **F**: vettore dei costi fissi relativi ad una location.

All'interno del codice sorgente del progetto una istanza di test viene modellata dalla classe **InstanceData_v2** nel modulo *benchmark*.
All'interno della classe sono state implementate le funzioni *encode* e *decode* che consentono di rappresentare matrici e vettori come stringhe binarie. Le due funzioni sono essenziali per le operazioni di salvataggio e caricamento dei dati dalla memoria di sistema.

Dati benchmark Durante la computazione di una istanza di test vengono raccolti diversi dati utili per le successive analisi prestazionali. Il risultato dell'esecuzione è la tupla

$$\langle id, c, l, I_{x,y,z}, R_{x,y,z}, L_{x,y,z}, sg_z, sg_{step}, I_t, R_t, L_t \rangle \tag{5}$$

dove i termini hanno il seguente significato:

- **id**: identificativo della istanza di test da cui è stato ottenuto il risultato
- **c** e **l**: rispettivamente il numero di clienti e di località
- $\mathbf{I_{x,y,z}}$: valori della soluzione ottima del modello intero
- $\mathbf{R_{x,y,z}}$: valori della soluzione ottima del modello con rilassamento del vincolo di interezza
- $\mathbf{L_{x,y,z}}$: valori della soluzione ottima del modello con rilassamento lagrangiano
- $\mathbf{sg_z}$: array contenente i valori della funzione obiettivo lagrangiana durante l'esecuzione dell'algoritmo del simplesso
- $\mathbf{sg_{step}}$: array contenente i valori del passo di ricerca durante l'esecuzione dell'algoritmo del simplesso
- $\mathbf{I_t}$: tempo di calcolo per il modello intero
- $\mathbf{R_t}$: tempo di calcolo per il modello con rilassamento intero
- $\mathbf{L_t}$: tempo di calcolo per il modello lagrangiano.

Processo di benchmark Il procedimento base utilizzato per l'esecuzione di una sessione di benchmark è riassunto dal diagramma sottostante.

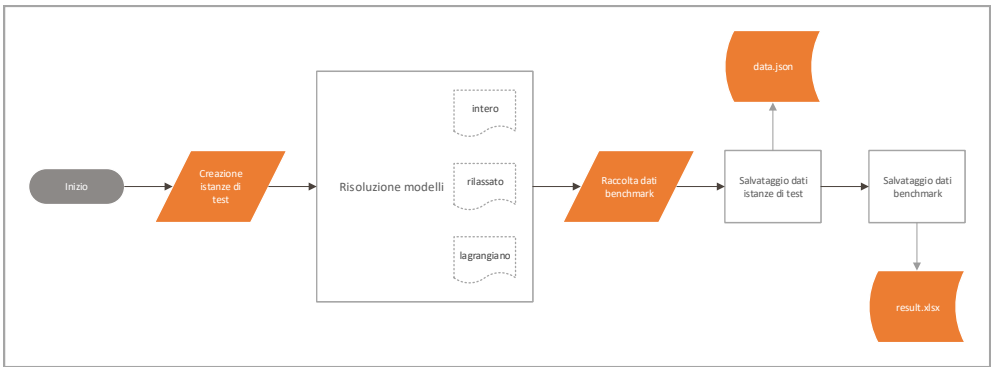


Figura 1: Versione *sequenziale*

La versione base del processo di benchmark prevede una esecuzione sequenziale di tutte le fasi:

1. Il primo passo consiste nella creazione delle istanze di test sulla base dei parametri di configurazione forniti alla funzione *run_benchmark* della classe **Runner**. I parametri da fornire alla funzione sono:
   - array:*shapes* contenente tutte le coppie *clienti* - *locations* per le quali generare una istanza
   - int:*count* indica il numero di istanze che devono essere generate a partire da una coppia
   - int:*jobs* indica il grado di parallelismo da utilizzare.
2. Create le istanze di test, esse vengono passate al modulo *solver_utils* che provvederà ad inizializzare e risolvere i tre modelli implementati (*intero, rilassamento intero, rilassamento lagrangiano*). Durante le operazioni di calcolo il modulo si occupa anche di registrare le informazioni che andranno poi a comporre il risultato della prova.
3. L'ultima fase del processo consiste nell'aggregare i risultati parziali delle singole istanze in un unico dataset (**DataFrame** di *pandas*), e successivamente memorizzare su disco una copia dei dati delle istanze (*data.json*) e del dataset finale (*result.xlsx*).

Parallelizzazione Al fine di ridurre il tempo totale necessario per il completamento di una sessione di benchmark è stata implementata una versione ad esecuzione parallela del processo base. Le modifiche hanno riguardato la fase di computazione delle istanze consentendo di eseguirne in parallelo.

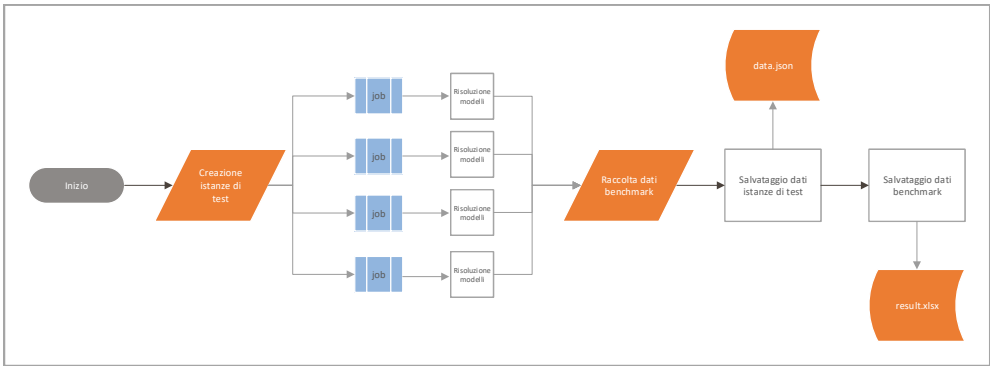


Figura 2: Versione *parallela*

Per l'implementazione è stato utilizzato il package *joblib*, il quale ha permesso di ridurre al minimo gli interventi sul codice originario. Lo *speedup* rilevato sulla macchina di test è stato di un fattore **x4** passando dalla versione sequenziale ($jobs = 1$) a quella parallela ($jobs = 8$).

6.2 Risultati

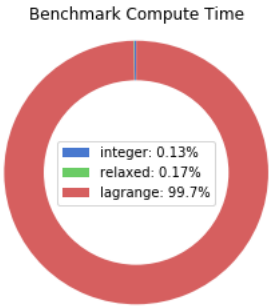
I risultati sono stati ottenuti utilizzando come parametri di inizializzazione:

- *clienti*: [10,20,40,60,80,120]
- *locations*: [2,4,8,16,24,32]
- *ripetizioni*: 10

I valori base sono stati poi combinati per formare **360 istanze** uniche.
L'intera sessione di benchmark è durata all'incirca **163 minuti**.

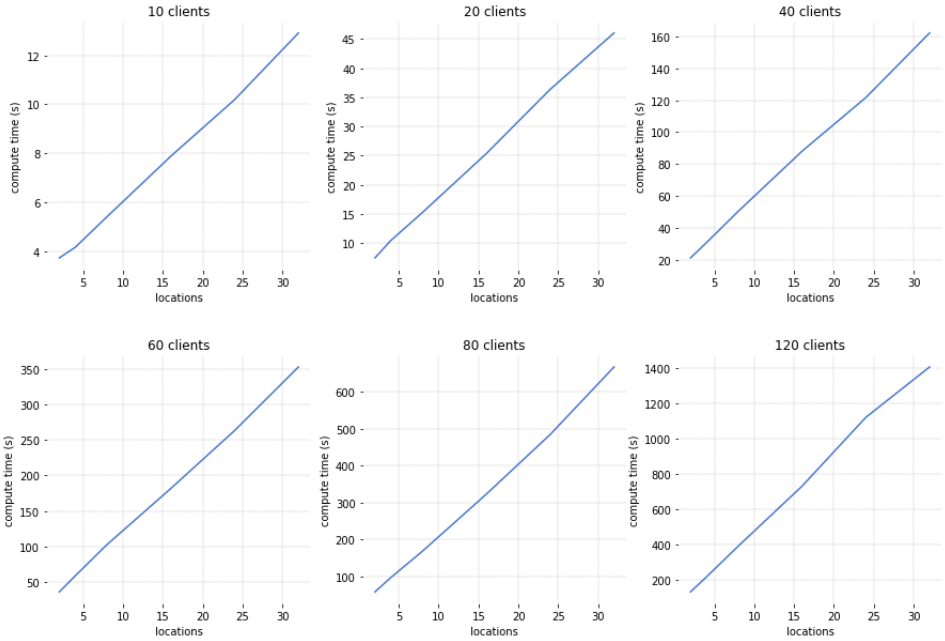
6.2.1 Tempo di calcolo

Durante l'esecuzione delle istanze di test si è tenuta traccia dei tempi di calcolo relativi a tutti e tre i modelli implementati. Una prima analisi mostra come il calcolo della soluzione per i modelli lagrangiani rappresenti la quasi totalità (**99.7%**) del tempo di calcolo complessivo per una sessione di benchmark.

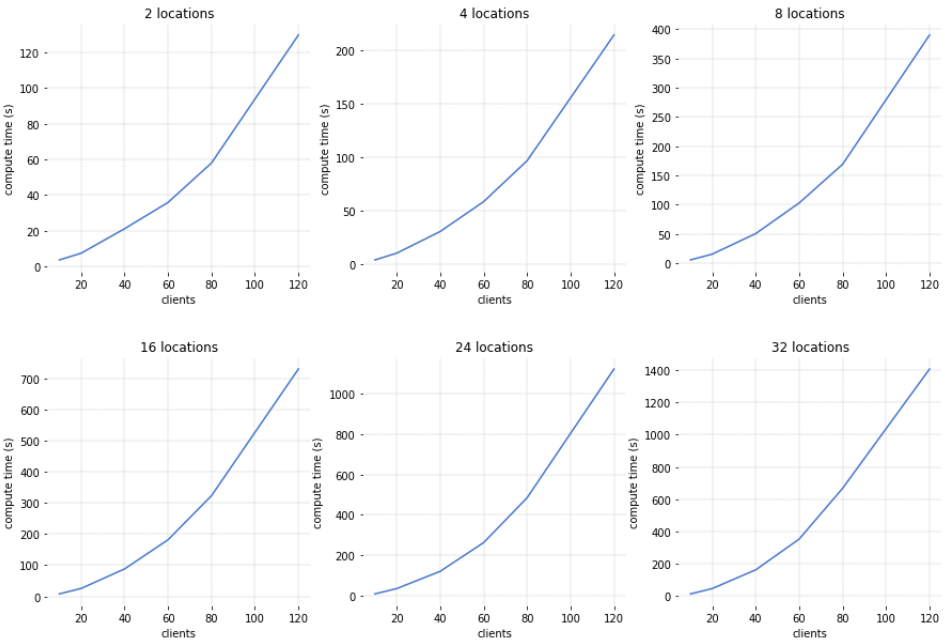


Nelle successive analisi ci si concentrerà esclusivamente su quest'ultimo modello.

Raggruppamento per numero di clienti Aggregando sulla base del numero di clienti da servire si può notare come il tempo di calcolo cresca proporzionalmente al numero di locations da valutare.

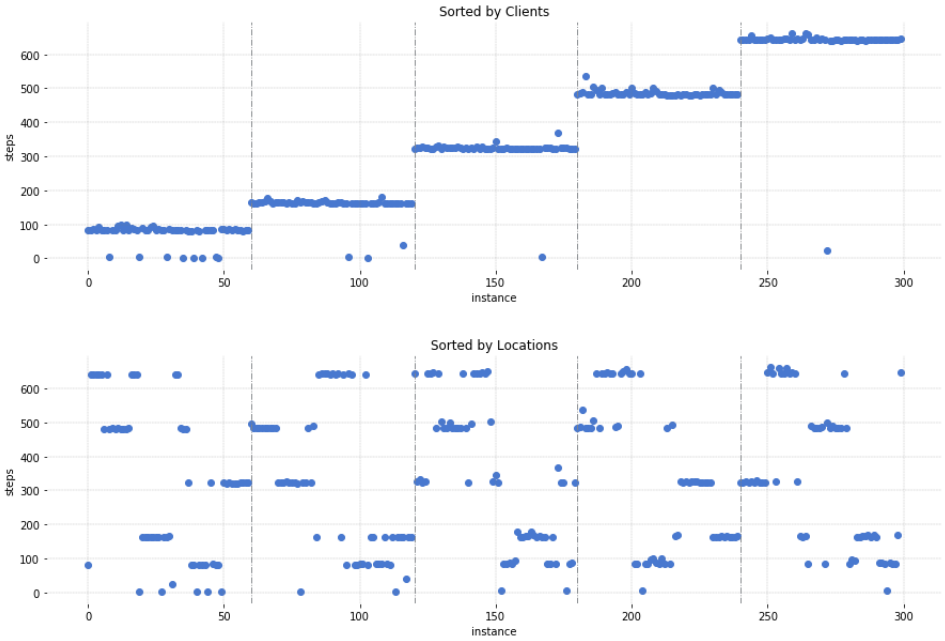


Raggruppamento per numero di località Aggregando invece sulla base del numero di località disponibili l'incremento del tempo di calcolo presenta un incremento quadratico rispetto al numero di clienti.



6.2.2 Velocità di convergenza

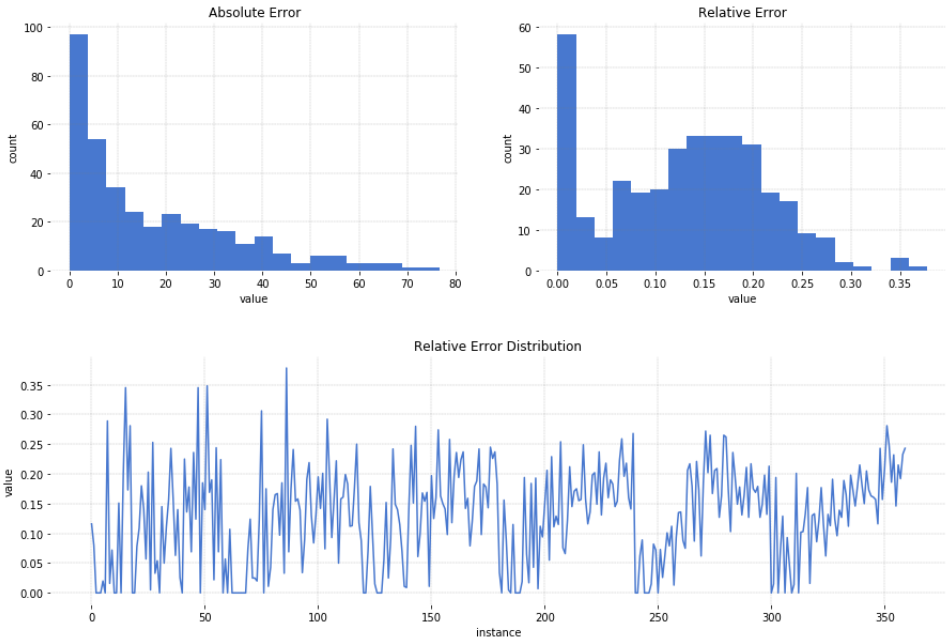
Studiando la velocità di convergenza dell'algoritmo del sottogradiente si è osservato che il numero di passi che l'algoritmo esegue prima di terminare è strettamente correlato con il numero di clienti. Viceversa non sembra esserci nessuna relazione con il numero di locations.



6.2.3 Errore di approssimazione

Osservando la distribuzione dell'errore di approssimazione commesso dall'algoritmo di sottogradiente si possono notare diverse cose:

1. In un numero rilevante di istanze la soluzione fornita dall'algoritmo è estremamente vicina alla soluzione ottima del problema primale
2. Il rapporto tra valore ottimo lagrangiano e valore intero si distribuisce intorno al +15% con un andamento apparentemente gaussiano
3. L'errore di approssimazione non presenta nessuna correlazione significativa con i due parametri di istanza principali.



Riferimenti bibliografici

[1] L.A. Wolsey. *Integer Programming*. Wiley Series in Discrete Mathematics and Optimization. Wiley, 1998.

[2] Gérard Cornuéjols, George L Nemhauser, and Lairemce A Wolsey. The uncapacitated facility location problem. Technical report, Carnegie-mellon univ pittsburgh pa management sciences research group, 1983.