Internet Routing

Federico Brandini Alessandro Mazzocchi

23 settembre 2024

1. Grafi e metriche

Un **grafo** G è una coppia (V, E), dove V è un insieme di nodi ed E è un insieme di archi, ossia coppie di nodi; se queste ultime sono ordinate, allora il grafo si dice **orientato**. In generale, se il grafo è non orientato vale che $(i, j) \in E \Rightarrow (j, i) \in E$, mentre non è necessariamente vero per un grafo orientato. Possiamo rappresentare un grafo anche attraverso la sua **matrice di adiacenza**, una matrice quadrata $\mathbf{A}$ di ordine $|V|$ tale che

$$\mathbf{A}(i,j) = \begin{cases} 1 & \text{se} \quad (i,j) \in E \\ 0 & \text{se} \quad (i,j) \notin E \end{cases} . \tag{1}$$

In particolare, se un grafo è non orientato, allora questo avrà matrice di adiacenza simmetrica. La matrice di adiacenza è uno strumento utile per ricercare un percorso, una sequenza di archi adiacenti, che consenta lo spostamento da un nodo di partenza a un nodo di arrivo. Se dovessero esistere più percorsi possibili, possiamo scegliere quello migliore estendendo la matrice di adiacenza assegnando ad ogni arco $(i, j) \in E$ il suo peso $\mathbf{A}(i, j)$ e per gli archi $(i, j) \notin E$ un valore neutro $\mathbf{A}(i, j)$ che non alteri i risultati e che in qualche modo escluda quei percorsi (falsi) che lo comprendono. Il problema da risolvere è ora un problema di ottimizzazione.

1.1 Shortest Path

Un esempio è il problema **Shortest Path**, per il quale si ricerca il percorso più breve da un nodo di partenza a uno di arrivo; in questo caso i valori di $\mathbf{A}(i, j)$ saranno le distanze $d_{ij} \geq 0$ fra tutte le possibili coppie di nodi $(i, j) \in E$ e $+\infty$ per le altre, ossia

$$\mathbf{A}(i,j) = \begin{cases} d_{ij} & \text{se} \quad (i,j) \in E \\ +\infty & \text{se} \quad (i,j) \notin E \end{cases} . \tag{2}$$

Il peso di un percorso p è la somma delle lunghezze (o distanze) di tutti gli archi che lo compongono. Indicando con $\mathcal{P}(i, j)$ l'insieme di tutti i percorsi dal nodo i al nodo j, il peso del percorso migliore da i a j è

$$\mathbf{A}^*(i,j) = \min_{p \in \mathcal{P}(i,j)} \sum_{(k,l) \in p} \mathbf{A}(k,l). \tag{3}$$

Notiamo che la metrica per la determinazione del peso del singolo percorso è definita tramite l'operatore + e che il criterio di determinazione del percorso ottimo avviene tramite l'operatore min.

1.2 Widest Path

Cambiando metrica e criterio, possiamo risolvere diversi tipi di problemi, come quello del **Widest Path**, per il quale si ricerca il percorso a banda più larga da un nodo di partenza a uno di arrivo; in questo caso i valori di $\mathbf{A}(i,j)$ saranno le larghezze di banda $b_{ij} \geq 0$ fra tutte le possibili coppie di nodi $(i,j) \in E$ e 0 per le altre, ossia

$$\mathbf{A}(i,j) = \begin{cases} b_{ij} & \text{se} \quad (i,j) \in E \\ 0 & \text{se} \quad (i,j) \notin E \end{cases} . \tag{4}$$

Il peso di un percorso p è il minimo fra le larghezze di banda di tutti gli archi che lo compongono. Indicando con $\mathcal{P}(i,j)$ l'insieme di tutti i percorsi dal nodo i al nodo j, il peso del percorso migliore da i a j è

$$\mathbf{A}^*(i,j) = \max_{p \in \mathcal{P}(i,j)} \min_{(k,l) \in p} \mathbf{A}(k,l). \tag{5}$$

Notiamo che la metrica per la determinazione del peso del singolo percorso è definita tramite l'operatore min e che il criterio di determinazione del percorso ottimo avviene tramite l'operatore max.

1.3 Most Reliable Path

L'ultimo esempio che trattiamo è quello del problema **Most Reliable Path**, per il quale si ricerca il percorso più affidabile da un nodo di partenza a uno di arrivo; in questo caso i valori di $\mathbf{A}(i,j)$ saranno i valori di affidabilità $r_{ij} \in [0,1]$ fra tutte le possibili coppie di nodi $(i,j) \in E$ e 0 per le altre, ossia

$$\mathbf{A}(i,j) = \begin{cases} r_{ij} & \text{se} \quad (i,j) \in E \\ 0 & \text{se} \quad (i,j) \notin E \end{cases} . \tag{6}$$

Il peso di un percorso p è il prodotto dei valori di affidabilità di tutti gli archi che lo compongono. Indicando con $\mathcal{P}(i,j)$ l'insieme di tutti i percorsi dal nodo i al nodo j, il peso del percorso migliore da i a j è

$$\mathbf{A}^*(i,j) = \max_{p \in \mathcal{P}(i,j)} \prod_{(k,l) \in p} \mathbf{A}(k,l). \tag{7}$$

Notiamo che la metrica per la determinazione del peso del singolo percorso è definita tramite l'operatore di moltiplicazione $\times$ e che il criterio di determinazione del percorso ottimo avviene tramite l'operatore max.

2. Thingest Path

Possiamo generalizzare i problemi precedenti al problema **Thingest Path**, per il quale si ricerca il percorso migliore rispetto a una certa metrica $\otimes$ per misurare i percorsi e a un criterio $\oplus$ per la determinazione di quello migliore. I valori di $\mathbf{A}(i,j)$ saranno i pesi $w_{ij} \in S$ per gli archi $(i,j) \in E$ e $\overline{0}$ per tutti gli altri, ossia

$$\mathbf{A}(i,j) = \begin{cases} w_{ij} & \text{se} \quad (i,j) \in E \\ \overline{0} & \text{se} \quad (i,j) \notin E \end{cases} , \tag{8}$$

dove $\overline{0}$ è elemento assorbente per $\otimes$ e neutro per $\oplus$. A questo punto, il peso del percorso migliore da i a j è

$$\mathbf{A}^*(i,j) = \bigoplus_{p \in \mathcal{P}(i,j)} \bigotimes_{(k,l) \in p} \mathbf{A}(k,l). \tag{9}$$

2.1 Algoritmo di Dijkstra Generalizzato

L'Algoritmo di Dijkstra nasce per risolvere il problema di Shortest Path e, in particolare, trova tutti i percorsi a peso minimo per andare da un nodo fissato i a un qualsiasi altro nodo del grafo. Per generalizzare tale algoritmo per un problema di Thingest Path, introduciamo la struttura matematica che ne fa da supporto.

Definizione

Una struttura $(S, \oplus, \otimes, \overline{0}, \overline{1})$ è un **semianello** se $\forall a, b, c \in S$

- $\oplus$ e $\otimes$ sono operazioni associative su S
- $\oplus$ è commutativa
- $\overline{0}$ è l'elemento neutro per $\oplus$, $a \oplus \overline{0} = \overline{0} \oplus a = a$
- $\overline{1}$ è l'elemento neutro per $\otimes$, $a \otimes \overline{1} = \overline{1} \otimes a = a$
- $\overline{0}$ è elemento assorbente per $\otimes$, $a \otimes \overline{0} = \overline{0} \otimes a = \overline{0}$
- vale la distributività a destra di $\otimes$ rispetto a $\oplus$, $a \otimes (b \oplus c) = (a \otimes b) \oplus (a \otimes c)$
- vale la distributività a sinistra di $\otimes$ rispetto a $\oplus$, $(a \oplus b) \otimes c = (a \otimes c) \oplus (b \otimes c)$.

Se inoltre $\overline{1}$ è elemento assorbente per $\oplus$, il semianello si dice **delimitato**. I semianelli per i problemi di Shortest Path, Widest Path e Most Reliable Path sono tutti delimitati e si trovano riassunti nella tabella sottostante.

problema	nome	S	$\oplus$	$\otimes$	$\overline{0}$	$\overline{1}$
Shortest Path	sp	$\mathbb{R}_0^+ \cup \{+\infty\}$	min	$+$	$+\infty$	0
Widest Path	bw	$\mathbb{R}_0^+ \cup \{+\infty\}$	max	min	0	$+\infty$
Most Realiable Path	rel	$[0,1]$	max	$\times$	0	1

Possiamo definire anche una relazione d'ordine su S basata sull'operatore $\oplus$; la definizione più naturale che si può dare è la seguente.

Definizione

Dati $a, b \in S$, scriviamo che

$$a \leq_\oplus b \tag{10}$$

se $a = a \oplus b$. Se inoltre $a \neq b$, diciamo anche che la disuguaglianza vale **in senso stretto** e scriviamo $a <_\oplus b$.

Se è sempre vero che $a \leq_\oplus b \vee b \leq_\oplus a$, allora la relazione d'ordine $\leq_\oplus$ si dice **totale**, altrimenti **parziale**. Nonostante esserlo per le tre istanze precedenti, in generale, tale relazione non è totale per qualunque semianello. Prima di annunciare l'algoritmo, è necessario dare un'ulteriore definizione relativa a un'operazione fra percorsi. Sia $\text{epaths}(E)$ l'insieme di tutti i percorsi aciclici su E.

Definizione

Siano $X, Y \subseteq \text{epaths}(E)$. La **concatenazione in coppia** (*pair-wise concatenation*) fra X e Y è l'insieme

$$X \diamond Y := \{pq \in \text{epaths}(E) \mid p \in X, q \in Y\}. \tag{11}$$

$X \diamond Y$ è l'insieme di tutti i percorsi aciclici ottenuti dalla concatenazione dei percorsi di X con quelli di Y. Indichiamo con ε il **percorso nullo**, null'altro che l'elemento neutro della composizione semplice fra percorsi contigui; cioè, per ogni percorso p vale che

$$p\varepsilon = \varepsilon p = p. \tag{12}$$

Inoltre, si potrebbe definire una struttura a semianello di percorsi aciclici nel modo seguente, dove $\wp(\cdot)$ indica l'insieme delle parti dell'insieme ricevuto per argomento.

nome	S	$\oplus$	$\otimes$	$\overline{0}$	$\overline{1}$
sep	$\wp(\text{epaths}(E))$	$\cup$	$\diamond$	$\{\}$	$\{\varepsilon\}$

Sulla base delle tre definizioni precedenti, possiamo generalizzare l'Algoritmo di Dijkstra in questo modo. Se $n = |V|$, i risultati del calcolo comprendono la matrice dei pesi ottimi

Algorithm 1 Algoritmo di Dijkstra Generalizzato

```
1:  procedure GeneralizedDijkstra(⊕, ⊗, 0̄, 1̄, A, i)
2:      for q ∈ V do
3:          D_0(i, q) ← 0̄; π_0(i, q) ← {}
4:      end for
5:      S_0 ← {}; D_0(i, i) ← 1̄; π_0(i, i) ← {ε}
6:      for k = 1, 2, ..., |V| do
7:          determinare q_k ∈ V \ S_{k−1} che minimizzi D_{k−1}(i, q_k) secondo ≤_⊕
8:          S_k ← S_{k−1} ∪ {q_k}
9:          for j ∈ V \ S_k do
10:             if D_{k−1}(i, q_k) ⊗ A(q_k, j) = D_{k−1}(i, j) then
11:                 π_k(i, j) ← π_k(i, j) ∪ (π_{k−1}(i, q_k) ⋄ {(q_k, j)})
12:             else if D_{k−1}(i, q_k) ⊗ A(q_k, j) <_⊕ D_{k−1}(i, j) then
13:                 π_k(i, j) ← π_{k−1}(i, q_k) ⋄ {(q_k, j)}
14:             end if
15:         end for
16:     end for
17: end procedure
```

$\mathbf{D} := \mathbf{D}_{n-1}$ e la matrice dei percorsi ottimi $\pi := \pi_{n-1}$. L'algoritmo calcola i percorsi

migliori dal nodo i a tutti gli altri nodi del grafo, cioè solo i valori delle i-esime righe di $\mathbf{D}$ e π. La complessità computazionale temporale dell'algoritmo è data dai due cicli annidati che iniziano a riga 6; il primo, quello più esterno, è eseguito esattamente n volte, mentre quello più interno (riga 9) al più n volte. Siccome la complessità del corpo di quest'ultimo è $O(n)$, **la complessità dell'Algoritmo di Dijkstra Generalizzato è $O(n^4)$**. In questa analisi abbiamo supposto costanti le complessità degli operatori $\oplus$ e $\otimes$ e che il grafo sia sparso (una rete, appunto), ossia un grafo $G = (V, E)$ tale che $|E| = O(n)$.

2.2 Prodotto Lessicografico

Date due metriche semplici, ci chiediamo se è possibile comporle per poterne realizzare una più complessa, che tenga conto di entrambe. Per esempio, potremmo voler scegliere fra i percorsi più brevi, quello a larghezza di banda più elevata, senza dover confrontare "manualmente" tutti i percorsi, ottenuti dall'algoritmo con metrica sp, con la metrica bw. L'operazione che permette di comporre due metriche è chiamata **prodotto lessicografico**.

Definizione

Dati due semianelli $s_1 = (S_1, \oplus_1, \otimes_1, \overline{0}_1, \overline{1}_1)$ e $s_2 = (S_2, \oplus_2, \otimes_2, \overline{0}_2, \overline{1}_2)$, il **prodotto lessicografico** fra s_1 e s_2 è la struttura algebrica

$$s_1 \vec{\times} s_2 := (S, \oplus, \otimes, \overline{0}, \overline{1}) \tag{13}$$

tale che

- $S := S_1 \times S_2$, dove $\times$ qui indica il prodotto cartesiano fra insiemi
- $(a, b) \otimes (c, d) := (a \otimes_1 c, b \otimes_2 d)$
- $(a, b) \oplus (c, d) := \begin{cases} (a, b) & \text{se } a <_{\oplus_1} c \\ (c, d) & \text{se } c <_{\oplus_1} a \\ (a, b \oplus_2 d) & \text{altrimenti} \end{cases}$ (*confronto lessicografico*)
- $\overline{0} := (\overline{0}_1, \overline{0}_2)$
- $\overline{1} := (\overline{1}_1, \overline{1}_2)$.

In generale, il prodotto lessicografico fra semianelli non è necessariamente un semianello; infatti, la proprietà che si perde facilmente da questa operazione è la distributività. Questa mancanza non garantisce più l'ottimalità dell'Algoritmo di Dijkstra Generalizzato, definito per semianelli. Tuttavia, ci viene in soccorso il seguente teorema, che ci aiuta a riconoscere quando i prodotti lessicografici danno vita a nuovi a semianelli e quando no.

Teorema

Il prodotto lessicografico fra due semianelli con rispettive metriche $\otimes_1$ e $\otimes_2$ è un semianello se e solo se $\otimes_1$ è cancellativa o $\otimes_2$ è costante.

La metrica $\otimes_1$ si dice **cancellativa** su S_1 se $\forall a, b, c \in S_1$

- $a \otimes_1 b = a \otimes_1 c \Rightarrow b = c$ (*cancellatività a sinistra*)
- $b \otimes_1 a = c \otimes_1 a \Rightarrow b = c$ (*cancellatività a destra*).

La metrica $\otimes_2$ si dice **costante** su S_2 se $\forall a, b, c \in S_2$

- $a \otimes_2 b = a \otimes_2 c$ (*costanza a sinistra*)
- $b \otimes_2 a = c \otimes_2 a$ (*costanza a destra*).

Possiamo utilizzare questo risultato per verificare alcuni prodotti lessicografici fra le metriche definite in precedenza.

problema	nome	definizione	semianello?
Most Reliable Shortest Path	mrsp	$\mathrm{sp} \vec{\times} \mathrm{rel}$	Sì
Widest Shortest Path	wsp	$\mathrm{sp} \vec{\times} \mathrm{bw}$	Sì
Shortest Widest Path	swp	$\mathrm{bw} \vec{\times} \mathrm{sp}$	No

Come si vede dalla tabella, non si hanno garanzie di ottimalità per il problema Shortest Widest Path. In questo caso conviene in primo luogo risolvere il problema Widest Path per poi procedere iterativamente alla selezione del percorso più breve fra quelli a banda più larga, aggiungendo complessità a valle.

3. Algoritmo di Routing e Parallelizzazione

Vogliamo realizzare un algoritmo che calcoli i percorsi ottimi per una qualsiasi coppia di nodi in un grafo. Lo chiameremo Algoritmo di Routing e può essere facilmente ottenuto eseguendo l'Algoritmo di Dijkstra Generalizzato su tutti i nodi i del grafo, ottenendo così la popolazione completa delle matrici $\mathbf{D}$ e π. La complessità temporale dell'Algoritmo di

Algorithm 2 Algoritmo di Routing

```
1: procedure ROUTING(⊕, ⊗, 0̄, 1̄, A)
2:     for i ∈ V do
3:         GENERALIZEDDIJKSTRA(⊕, ⊗, 0̄, 1̄, A, i)
4:     end for
5: end procedure
```

Routing è $O(n^5)$. Notiamo che le iterazioni del corpo del ciclo a riga 2 sono totalmente idipendenti fra loro; questo ci consente di eseguirle in parallelo, applicando il pattern *Embarrassingly Parallel* (abbiamo soltanto letture concorrenti su $\mathbf{A}(i, j)$, che non causano corse critiche). Se analizziamo l'algoritmo parallelo con il modello DAG (*Direct Acyclic Graph*), un modello *Work-Depth*, abbiamo un lavoro totale

$$W(n) = O(n^5), \tag{14}$$

una profondità

$$D(n) = O(n^4) \tag{15}$$

e, quindi, un **parallelismo**

$$P(n) = \frac{W(n)}{D(n)} = \boldsymbol{O(n)}, \tag{16}$$

che è lineare. Inoltre, siccome

$$\lim_{n \to +\infty} P(n) = +\infty, \tag{17}$$

l'Algoritmo di Routing è **scalabilmente parallelo**.

4. Implementazione

Per implementare le strutture matematiche discusse nella prima parte, scegliamo come linguaggio di programmazione C++. Questa scelta ci consente di creare una struttura generale e allo stesso tempo parallela grazie alla libreria OpenMP (*Open MultiProcessing*). Siccome OpenMP lavora sui core della CPU, il livello effettivo di scalabilità parallela offerto dalla libreria è di per sé limitato; per questa ragione, decidiamo di implementare anche una versione meno generale, ma molto più scalabile, utilizzando i core della GPU con CUDA (*Compute Unified Device Architecture*), questa volta in C. Nella tabella sottostante riassumiamo i risultati per i test temporali effettuati sulle tre versioni dell'algoritmo (sequenziale (integrata in quella con OpenMP), parallela con OpenMP, parallela con CUDA) per un'istanza di problema Shortest Widest Path con 149 nodi (rete Colt Telecom). Le esecuzioni delle prime due hanno avuto luogo sullo stesso calcolatore (macchina virtuale con 4 core), mentre la terza sul cluster dell'Università di Parma, che dispone, fra le varie, di una GPU nvidia con 6912 core.

#nodi	T_{seq}	T_{OpenMP}	T_{CUDA}
149	≈ 4 s	≈ 1 s	≈ 1 s

Qualsiasi considerazione sui tempi deve tenere conto dell'overhead parallelo.

5. Conclusioni e Sviluppi futuri

L'obiettivo di questo progetto sta nell'implementazione **parallela** dell'Algoritmo di Routing, mantenendo allo stesso tempo **generale** la struttura algebrica a semianello sottostante, consentendo all'utente programmatore di definire la metrica e il criterio per la scelta dei percorsi ottimi. La versione con OpenMP è molto generale ma con scalabilità parallela limitata, mentre la versione CUDA è meno generale ma altamente scalabile. La prosecuzione del progetto potrebbe partire proprio da qui, cercando di rendere generale anche quest'ultima versione, magari estendendo la versione di C in C++. Un'idea potrebbe essere quella di fornire a un qualsiasi oggetto presente nella funzione kernel due metodi: uno per lo spostamento da memoria Host a memoria Device e uno per lo spostamento inverso con liberazione della memoria allocata sul Device. Questo si traduce nell'imporre ai tipi interessati un vincolo relativo all'implementazione di un'interfaccia contenente la dichiarazione dei due metodi precedenti; un nome per tale interfaccia potrebbe essere "CUDAble" in riferimento alla nomenclatura adottata. Qualora fosse possibile, per particolari semplici casi si potrebbe sfruttare anche la gestione unificata della memoria.

Riferimenti bibliografici

Dynerowicz S., Griffin T. G. (2013). “On the Forwarding Paths Produced by Internet Routing Algorithms”. In.