Università degli Studi di Udine

Dipartimento di Scienze Matematica, Informatiche e Fisiche

Tesi Magistrale in Informatica

Algoritmi e Ragionamento Automatico

Uno studio sugli algoritmi di sketching per la stima della cardinalità

Relatore
Prof. Gabriele Puppis
Università degli Studi di Udine

Laureando Magistrale
Daniele Ferroli

Matricola
137357

Anno Accademico
2024-2025

"Da scrivere"
— Rene Descartes

Contents

Notazioni e concetti preliminari

In questa sezione si raccolgono le notazioni utilizzate nel resto della tesi.

- $\mathcal{U}$: universo degli elementi.
- $S = \langle x_1, \ldots, x_s \rangle$: stream di dati.
- $n = |\mathcal{U}|$: dimensione dell'universo.
- $f(a)$: frequenza di $a \in \mathcal{U}$.
- F_k: frequency moments.
- F_0: numero di distinti nella stream.
- $\hat{F}_0$: stima di F_0 prodotta da un algoritmo.
- (ε, δ): parametri di accuratezza; ε è l'errore relativo ammesso e $1 - \delta$ è la probabilità di successo.
- m: memoria dello sketch (in bit o in byte).
- M: stato interno dell'algoritmo di streaming (lo sketch).

1 Introduzione

Random citation [**?**].

2 Background

In questa tesi, il modello che rappresenta i dati in input, a differenza di algoritmi più tradizionali, è chiamato *modello di stream di dati* (data stream model).

2.1 Modello di stream di dati

Nel modello di stream i dati [1] arrivano in modo continuo e potenzialmente infinita. Rispetto l'utilizzo di un database tradizionale, non è possibile accumulare tutto in memoria o su disco e interrogare i dati. Gli elementi devono essere processati al volo oppure vengono persi.

Inoltre, la velocità con cui i dati arrivano non è controllata dal sistema (più stream possono arrivare a velocità e con formati diversi) e lo spazio di memoria disponibile è limitato. Eventuali archivi storici possono esistere, ma non sono pensati per rispondere a query online in tempi ragionevoli.

Iniziamo a definire formalmente gli elementi di una stream e come vengono processati.

Def. 2.1 (Stream di dati)**.** Sia $\mathcal{U}$ un universo di chiavi. Senza perdita di generalità, assumiamo $\mathcal{U} \subseteq \mathbb{N}$. Una *stream di dati* è una sequenza ordinata di elementi

$$S = \langle x_1, x_2, \ldots, x_s \rangle,$$

dove ogni $x_i \in \mathcal{U}$ e s può essere molto grande o non noto a priori.

Per analizzare una stream è utile descriverla tramite le frequenze degli elementi.

Def. 2.2 (Frequenze)**.** Data una stream S, la *frequenza* di un elemento $a \in \mathcal{U}$ è

$$f(a) = |\{i \mid x_i = a\}|.$$

La collezione delle frequenze può essere vista come un vettore $f \in \mathbb{N}^{|\mathcal{U}|}$.

Una volta definita la nozione di frequenza, si specifica il modello di aggiornamento con cui la stream viene osservata. Nel modello della stream dei dati esistono diverse tipologie di modelli [2]. In questa tesi adottiamo il seguente.

Def. 2.3 (Modello *insertion-only*)**.** La stream è una sequenza di aggiornamenti del tipo (a_t, Δ_t), con $a_t \in \mathcal{U}$ e $\Delta_t \geq 0$. Indichiamo con $A_t[j]$ la frequenza dell'elemento j dopo i primi t aggiornamenti; allora

$$A_t[j] = \begin{cases} A_{t-1}[j] + \Delta_t & \text{se } a_t = j, \\ A_{t-1}[j] & \text{altrimenti.} \end{cases}$$

Se la stream è una lista di valori, ogni elemento x_i può essere visto come un aggiornamento $(x_i, 1)$.

Esistono tuttavia modelli più generali. Nel *turnstile* sono ammessi anche aggiornamenti negativi, così che le frequenze possano aumentare o diminuire. Nel modello a *sliding window* si considerano solo gli ultimi W aggiornamenti della stream, scartando i più vecchi. Questi casi esulano dallo scopo della tesi, ma sono citati per completezza.

Fissato il modello, l'obiettivo principale della tesi è stimare la cardinalità dell'insieme dei distinti.

Def. 2.4 (Numero di distinti)**.** Il *numero di distinti* nella stream S è

$$F_0 = |\{a \in \mathcal{U} \mid f(a) > 0\}|.$$

Più in generale, il numero di distinti è un caso particolare di una famiglia di misure note come *frequency moments*.

Def. 2.5 (Frequency moments)**.** Per ogni $k \geq 0$, il *frequency moment* F_k è definito come

$$F_k = \sum_{a \in \mathcal{U}} f(a)^k.$$

In particolare, F_0 corrisponde al numero di distinti.

Per valutare la qualità di una stima si introduce la nozione di approssimazione con parametri di accuratezza e confidenza.

Def. 2.6 ((ε, δ)-approssimazione)**.** Un algoritmo A è detto (ε, δ)-*approssimante* per F_0 se, per ogni stream, produce una stima $\tilde{F}_0$ tale che

$$\Pr\big(|\tilde{F}_0 - F_0| \leq \varepsilon F_0\big) \geq 1 - \delta,$$

dove la probabilità è rispetto alla randomizzazione interna dell'algoritmo [3].

Esempio. Con $\varepsilon = 0{,}05$ e $\delta = 0{,}01$, l'algoritmo deve restituire una stima entro il 5% da F_0 con probabilità almeno 99%.

Supponiamo inoltre che l'universo abbia dimensione $n = |\mathcal{U}|$ e che ogni elemento x_i richieda b bit per essere rappresentato.

Le garanzie di accuratezza devono convivere con vincoli stringenti di tempo e memoria. In questo contesto, un algoritmo di streaming deve:

- processare ciascun elemento con costo $O(1)$ o quasi costante;
- usare memoria molto più piccola di n e di $|\mathcal{U}|$;
- produrre una stima $\hat{F}_0$ con errore controllato, utilizzando m bits, dove $m \ll n$.

Per rispettare questi vincoli si ricorre a funzioni hash che approssimano una distribuzione uniforme sugli elementi e a strutture compatte, chiamate **sketch**, che riassumono le informazioni essenziali della stream senza conservarla esplicitamente.

Il vincolo più forte è quello di memoria: si richiede che lo spazio cresca molto più lentamente della dimensione dell'universo.

Def. 2.7 (Spazio sublineare)**.** Un algoritmo usa *spazio sublineare* se la memoria m cresce asintoticamente meno di n, cioè $m = o(n)$, dove $n = |\mathcal{U}|$. Si richiede che m dipenda in modo polilogaritmico da n e polinomiale da $1/\varepsilon$ e $\log(1/\delta)$.

Esempio. Per il problema dei distinti esistono algoritmi che ottengono una $(1 \pm \varepsilon)$-approssimazione usando $O(\varepsilon^{-2} + \log n)$ bit [4], che è molto meno dei $\Omega(n)$ bit necessari per memorizzare l'insieme dei distinti.

2.2 Algoritmi di streaming

Il modello di data stream con le caratteristiche appena descritte, flussi potenzialmente infiniti, velocità non controllata e memoria limitata, rendono inapplicabili gli approcci classici basati su memorizzazione completa e analisi a posteriori.

Di conseguenza gli algoritmi di streaming nascono quindi per produrre stime e statistiche utili durante l'arrivo dei dati, lavorando in un solo passaggio e mantenendo una sintesi compatta della stream.

Def. 2.8 (Algoritmo di streaming)**.** Un algoritmo di streaming elabora una stream in un solo passaggio e mantiene uno stato interno M di dimensione limitata m. Per ogni elemento x_i della stream, lo stato viene aggiornato tramite una funzione

$$M \leftarrow \mathrm{Update}(M, x_i),$$

e in qualunque momento è possibile ottenere una risposta (o stima) tramite

$$\hat{F}_0 \leftarrow \mathrm{Query}(M).$$

Nel modello classico si richiede che m sia sublineare rispetto a n e che il tempo per aggiornamento sia $O(1)$ o quasi costante [2].

Dopo ogni aggiornamento, l'elemento appena osservato può essere scartato: lo stato M rappresenta una sintesi compatta dei dati, spesso chiamata *sketch* [5].

Nel modello classico, la qualità di un algoritmo di streaming si valuta tipicamente in termini di numero di passaggi sulla stream, della quantità di memoria usata, del tempo necessario per processare un elemento e della precisione della risposta. Per algoritmi di approssimazione, questa precisione è descritta da un rapporto di approssimazione e da una probabilità di successo, spesso espressi tramite il modello (ε, δ) [6].

Gli algoritmi di streaming sono affini agli algoritmi *online*[7], perché operano senza disporre dell'intero input; tuttavia non sono identici, poiché nel modello streaming è possibile talvolta differire l'azione fino all'arrivo di piccoli blocchi di elementi, pur mantenendo una memoria molto limitata [2, 6]. Un possibile esempio è fornito dagli algoritmi per la *sliding window*, che mantengono riassunti a blocchi per stimare statistiche recenti con memoria limitata [8]. Nel nostro contesto, tutti gli algoritmi che sono stati implementati e trattati sono algoritmi anche online, perché processano ogni elemento appena arriva, senza differire l'azione.

Per il problema dei distinti, la letteratura evidenzia un legame diretto tra accuratezza e spazio: ridurre ε implica un incremento della memoria necessaria. In particolare, si considerano efficienti gli algoritmi che usano solo spazio polinomiale in $1/\varepsilon$ e logaritmico nella lunghezza della stream e nella dimensione dell'universo, con un costo per elemento molto basso [3]. Questo mette in evidenza il trade-off centrale tra precisione e spazio necessario dello sketch.

Questi algoritmi sono spesso randomizzati: la probabilità nella definizione di (ε, δ) è rispetto alle scelte casuali interne dell'algoritmo, e rappresenta una garanzia probabilistica sulla qualità della stima [3]. Nelle implementazioni pratiche, questa randomizzazione è tipicamente incarnata dalla funzione di hash, assunta sufficientemente vicina a una scelta casuale (o parametrizzata da un seed). La randomizzazione consente di ridurre drasticamente lo spazio rispetto alle soluzioni deterministiche, a patto di accettare un errore controllato con alta probabilità.

Un'altra differenza rilevante rispetto all'analisi tradizionale è la distinzione tra stime in tempo reale e analisi offline. Nei sistemi classici, gli aggiornamenti si registrano in un archivio e le analisi complesse vengono svolte offline in warehouse. Nel modello di streaming, invece, molte applicazioni richiedono elaborazioni sofisticate in quasi tempo reale, come rilevamento di anomalie, monitoraggio di trend o cambiamenti improvvisi e questo condiziona la progettazione degli algoritmi [2].

Infine, in contesti distribuiti è spesso necessario combinare riassunti di porzioni diverse delle streams. Il concetto di *mergeabilità* formalizza la possibilità di unire due sintesi in una sintesi della loro unione preservando le garanzie di errore e la dimensione dello stato: questo permette di scalare gli algoritmi a scenari paralleli o gerarchici ed è una proprietà centrale per gli sketch moderni [9].

2.3 Funzioni hash

Qui si introduce il ruolo delle funzioni hash nella riduzione dello spazio e nella randomizzazione delle stime. Da completare:

- Definizione di hash e proprietà desiderate.
- Modello ideale: $h : \mathcal{U} \to [0, 1)$ o $h : \mathcal{U} \to \{0, 1\}^w$.
- Assunzioni tipiche: uniformità, indipendenza (o hash quasi-uniformi).
- Impatto di scelte non ideali sull'errore degli stimatori.
- Eventuale gestione del seed per riproducibilità.

2.4 Sketch

Si definisce lo sketch come struttura compatta che riassume la stream. Da completare:

- Definizione generale di sketch e operazioni supportate (update, query).
- Dimensione dello sketch e relazione con i parametri (k, m, L).
- Esempi intuitivi (registri, bitmap, contatori).
- Due forme usate in tesi:
  - bitmap / pattern di bit (Probabilistic Counting).
  - registri e leading zeros (LogLog, HLL, HLL++).
- (Cenno) serializzabilità e footprint in byte.

2.5 Stimatori

Si descrive cosa si intende per stimatore e in che senso la stima è corretta. Da completare:

- Definizione di stimatore $\hat{F}_0$ e proprietà desiderate.
- Distinzione tra stimatore corretto (unbiased) e stimatore biased.
- Consistenza e varianza dello stimatore.
- Cenno a tecniche di bias correction.

2.6 Metriche di errore

Questa sezione introduce le metriche di qualità della stima. Da completare:

- Errore assoluto e relativo.
- Bias e unbiased estimator.
- Varianza e standard error.
- RMSE e MAE come metriche aggregate.
- RSE teorica vs osservata (quando applicabile).
- Definizioni allineate alle colonne CSV del framework.

2.7 Famiglia di algoritmi per count-distinct

Sezione ponte (senza dettagli implementativi) per motivare il Capitolo 3. Da completare:

- Baseline esatta vs sketch probabilistici.
- Linea FM/Probabilistic Counting $\rightarrow$ LogLog $\rightarrow$ HLL $\rightarrow$ HLL++.
- Differenze qualitative: riduzione varianza, correzioni di range, uso di registri.

2.8 Spazio–accuratezza: ordini di grandezza

Sezione opzionale ma utile per giustificare l'uso degli sketch. Da completare:

- Dipendenza dello spazio da (ε, δ).
- Interpretazione di "ottimalità" a livello di ordine di grandezza.

3

Un’analisi sullo stato dell’arte

TODO: scrivere l’analisi dello stato dell’arte.

4

Implementazione

4.1 Algoritmi

4.2 Framework di benchmark

Example of Algorithm 4.1 reference.

Algorithm 4.1 Pseudocode

```
i ← 10
if i ≥ 5
    i ← i − 1
else
    if i ≤ 3
        i ← i + 2
    end if
end if
```

5
Risultati sperimentali

Example of Table 5.1, made using `https://www.tablesgenerator.com/`.

A	B
1	2
3	4

Table 5.1: Interesting results.

6
Conclusioni

References

[1] J. Leskovec, A. Rajaraman, and J. D. Ullman, *Mining Data Streams*. Cambridge University Press, 2014, p. 123–153.

[2] S. Muthukrishnan, "Data streams: Algorithms and applications," Rutgers University, Tech. Rep., 2005.

[3] Z. Bar-Yossef, T. S. Jayram, R. Kumar, D. Sivakumar, and L. Trevisan, "Counting distinct elements in a data stream," in *Randomization and Approximation Techniques in Computer Science (RANDOM 2002)*, ser. Lecture Notes in Computer Science, vol. 2483. Springer, 2002, pp. 1–10.

[4] D. M. Kane, J. Nelson, and D. P. Woodruff, "An optimal algorithm for the distinct elements problem," in *Proceedings of the 29th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (PODS 2010)*. ACM, 2010.

[5] G. Cormode, "Data sketching," *Communications of the ACM*, 2017.

[6] N. Prezza, "Algorithms for massive data – lecture notes," 2025, lecture notes, Ca' Foscari University of Venice.

[7] R. M. Karp, "On-line algorithms versus off-line algorithms: How much is it worth to know the future?" in *IFIP Congress (1)*. World Computer Congress, 1992, pp. 416–429.

[8] M. Datar, A. Gionis, P. Indyk, and R. Motwani, "Maintaining stream statistics over sliding windows: (extended abstract)," in *Proceedings of the Thirteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, ser. SODA '02. USA: Society for Industrial and Applied Mathematics, 2002, pp. 635–644.

[9] P. K. Agarwal, G. Cormode, Z. Huang, J. M. Phillips, Z. Wei, and K. Yi, "Mergeable summaries," in *Proceedings of the 31st ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems (PODS 2012)*. ACM, 2012.

Acknowledgments