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).
- V: dominio di uscita di una funzione di hash.
- $h : \mathcal{U} \to V$: funzione di hash.
- $\mathcal{H}$: famiglia di funzioni di hash.
- w: numero di bit del valore hash (se $V = \{0, 1\}^w$).
- $\mathcal{S}$: spazio degli stati di uno sketch.
- $\oplus$: operatore di merge tra stati di 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—rende inapplicabili gli approcci classici basati su memorizzazione completa e analisi a posteriori.

Gli algoritmi di streaming nascono 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].

Metriche di valutazioni

Nel modello classico, le prestazioni si misurano in termini di **passaggi** sulla stream, **memoria** usata, **tempo per elemento** e **accuratezza** della risposta. Per algoritmi di approssimazione, l'accuratezza è espressa tramite un rapporto di approssimazione e una probabilità di successo, spesso nel modello (ε, δ) [6].

Confronto con gli algoritmi online

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 implementati e trattati sono anche *online*, perché processano ogni elemento appena arriva, senza differire l'azione.

Limitazioni di accuratezza rispetto allo spazio

Per il problema dei calcolo degli elementi 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.

Randomizzazione e garanzie probabilistiche

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.

Real-time vs offline

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 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].

Mergeability

In contesti distribuiti è spesso necessario combinare riassunti di porzioni diverse della stream. 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 di hash

Le funzioni hash sono il principale strumento per "randomizzare" lo stream e associare un universo molto grande in un dominio più piccolo, rendendo possibile l'uso di strutture compatte. In molti sketch, la qualità della stima dipende direttamente dalle proprietà della funzione di hash utilizzata [10, 11].

Def. 2.9 (Funzione di hash)**.** Una funzione di hash h è una funzione deterministica

$$h : \mathcal{U} \to V,$$

che associa a ogni chiave dell'universo $\mathcal{U}$ un valore in un dominio V di dimensione molto più piccola, tipicamente $V = \{0,1\}^w$ oppure $V = [0,1)$ tramite normalizzazione.

Quindi, una funzione di hash mappa dati di lunghezza arbitraria in un valore di lunghezza fissa, chiamato *hash value*, spesso usato per indicizzare delle strutture dati come le tabelle di hash.

Questa trasformazione permette accessi veloci e riduce lo spazio necessario rispetto alla memorizzazione diretta delle chiavi.

Poiché più chiavi possono produrre lo stesso valore, le *collisioni* sono inevitabili: una buona funzione di hash deve essere veloce da calcolare e minimizzare la probabilità di collisione, idealmente distribuendo i valori in modo uniforme sul dominio [12].

Proprietà desiderate. Le proprietà classiche richieste sono: l'**uniformità** (le chiavi sono distribuite in modo uniforme su V), la **bassa probabilità di collisione**, l'**indipendenza** tra le immagini di chiavi diverse e l'**efficienza** di calcolo.

Una funzione di hash con queste proprietà rende la stima degli sketch stabili e con varianza controllata [10, 11].

Def. 2.10 (Modello di hashing uniforme)**.** Nel modello ideale, h è scelta uniformemente a caso dall'insieme di tutte le funzioni $\mathcal{U} \to V$. In questo caso, per ogni chiave $x \in \mathcal{U}$, il valore $h(x)$ è uniforme in V e le immagini di chiavi distinte sono indipendenti [10, 11].

Def. 2.11 (Famiglia universale [13])**.** Una famiglia $\mathcal{H}$ di funzioni $\mathcal{U} \to V$ è *universale* se, per ogni coppia di chiavi distinte $x \neq y$, vale

$$\Pr_{h \leftarrow \mathcal{H}}\big[h(x) = h(y)\big] \leq \frac{1}{|V|}.$$

Def. 2.12 (k-wise indipendenza [10])**.** Una famiglia $\mathcal{H}$ è *k-wise indipendente* se, per ogni scelta di k chiavi distinte $x_1, \ldots, x_k$, il vettore

$$\big(h(x_1), \ldots, h(x_k)\big)$$

è distribuito uniformemente in V^k quando h è scelta a caso da $\mathcal{H}$.

Assunzioni tipiche. Nelle analisi teoriche si assume spesso il modello ideale di hashing uniforme; in alternativa si usa una famiglia con un grado limitato di indipendenza (ad esempio k-wise), o una famiglia universale [13, 10]. In pratica, l'uso di funzioni semplici è comune, ma le garanzie possono degradare rispetto al modello ideale se le assunzioni non sono soddisfatte.

Impatto delle scelte. Una funzione di hash non sufficientemente uniforme può introdurre collisioni sistematiche o correlazioni tra registri, con un aumento del bias e della varianza degli stimatori [10, 11].

2.4 Sketch

Uno *sketch* è una struttura dati probabilistica che riassume una stream attraverso uno stato M aggiornabile con operazioni *update* e interrogabile con operazioni *query*. Lo sketch non conserva gli elementi originali, ma solo le informazioni necessarie per stimare una quantità d'interesse con memoria limitata che deve essere molto inferiore rispetto a conservare i dati originali.

Merge-ability. Nei contesti distribuiti è utile poter combinare due sketch costruiti su porzioni diverse della stream. Come possibile esempio, si pensi a due server che creano in maniera indipendente la loro sketch dei dati e che si voglia unire queste informazioni. Intuitivamente, ciò è possibile solo se gli sketch condividono gli stessi parametri strutturali e la stessa funzione di hash (o lo stesso seed), altrimenti la fusione può degradare le garanzie.

Def. 2.13 (Sketch mergeable)**.** Sia $S(\cdot)$ la procedura che costruisce uno sketch e sia $\oplus$ un operatore sullo stato. Lo sketch è *mergeable* se, per due dataset D_1, D_2 costruiti con la stessa parametrizzazione e la stessa hash, vale

$$S(D_1) \oplus S(D_2) \approx S(D_1 \cup D_2),$$

preservando le garanzie di errore (o con degradazione nota) [9].

Def. 2.14 (Operatore chiuso)**.** Sia $\mathcal{S}$ lo spazio degli stati di uno sketch. Un operatore $\oplus$ è *chiuso* se

$$\oplus : \mathcal{S} \times \mathcal{S} \to \mathcal{S},$$

cioè se la combinazione di due stati produce ancora uno stato valido dello stesso tipo.

In pratica, l'operatore $\oplus$ deve essere *chiuso* sullo stato (ad esempio utilizzando la funzione di massimo per registro o la funzione di somma per componente) e, per aggregazioni robuste, è preferibile che sia *commutativo* e *associativo*; in molti casi è utile anche l'*idempotenza* per tollerare duplicazioni.

Osservazione: Per gli algoritmi che vedremo in seguito, se due sketch hanno gli stessi parametri e la stessa funzione di hash/seed, lo sketch che si ottiene "unendo" registro per registro è equivalente allo sketch costruito sulla concatenazione (o unione) delle due stream.

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.

[10] S. Vadhan and M. Mitzenmacher, "Why simple hash functions work: Exploiting the entropy in a data stream," 2007, manuscript.

[11] A. Pagh and R. Pagh, "Uniform hashing in constant time and optimal space," 2007, manuscript.

[12] "Hash function," Wikipedia, 2026, accessed 2026-02-07. [Online]. Available: https://en.wikipedia.org/wiki/Hash_function

[13] J. L. Carter and M. N. Wegman, "Universal classes of hash functions," *Journal of Computer and System Sciences*, vol. 18, pp. 143–154, 1979.

Acknowledgments