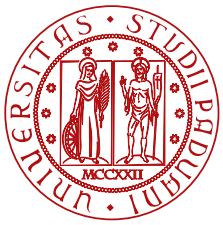

UNIVERSITÀ DEGLI STUDI DI PADOVA

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

CORSO DI LAUREA IN INGEGNERIA INFORMATICA

Progetto e sperimentazione di un algoritmo per Fair K-center Clustering nel modello sliding window

Relatori
Prof. Ceccarello Matteo
Prof. Pietracaprina Andrea
Prof. Pucci Geppino

Laureando
Visona' Francesco
Matricola 2032393

ANNO ACCADEMICO 2023-2024

Data di laurea 27/09/2024

Ai miei genitori e ai miei fratelli,
che con affetto mi hanno ispirato a diventare la persona che sono

Sommario

L'obiettivo di questo lavoro è di misurare le prestazioni dell'algoritmo di Cappellotto e al. [1] per la risoluzione di un problema di Fair K-Center nel modello Sliding Window, con particolare riguardo rispetto all'attuale stato dell'arte sviluppato da Chen e al. [2]. Mentre il problema di K-Center Clustering chiede di trovare, dato un insieme di punti presi da uno spazio metrico e un intero K, un sottoinsieme di centri tale per cui la distanza massima di un punto dall'insieme di centri sia minimizzata, nel problema FKC si ha il vincolo aggiuntivo che tale sottoinsieme deve essere un insieme indipendente per un matroide di partizione. Questo matroide, che può essere utilizzato per modellare vincoli di equità, è applicabile quando i punti di W sono divisibili in $\ell \leq K$ gruppi, nel qual caso un insieme indipendente include al più k_i punti per il gruppo i, con k_i tali che la loro somma faccia K. Nel modello di calcolo Sliding Window in qualsiasi momento l'algoritmo deve essere in grado di calcolare in modo efficiente una soluzione a tale problema per l'insieme degli ultimi N punti dello stream. L'algoritmo di Cappellotto e al., il primo algoritmo che sia progettato specificatamente per tale modello, presenta un fattore di approssimazione pari a $3 + \epsilon$, dove 3 è la migliore approssimazione ottenibile sequenzialmente in tempo polinomiale e $0 < \epsilon < 1$ rappresenta un parametro di precisione definito dall'utente. Dimostreremo in questa tesi che tale algoritmo è sperimentalmente migliore in gran parte dei casi di utilizzo rispetto allo stato dell'arte, ed è altamente semplice da scalare per soddisfare necessità di prestazioni temporali e spaziali o di qualità.

Indice

Elenco delle figure

Capitolo 1

Introduzione

Negli ultimi anni è aumentata l'attenzione nei confronti dell'apprendimento non supervisionato, che prevede di ottenere dati da un dataset che non sia stato già fornito di etichette. In questo contesto, sono molto importanti i problemi di clustering: tali problemi hanno lo scopo di ottenere una partizione di un insieme di oggetti (rappresentati come punti di uno spazio metrico, come ad esempio lo spazio Euclideo) in cluster di oggetti simili tra loro più di quanto lo siano con gli oggetti degli altri cluster. Tale analisi si dimostra veramente utile in una grande varietà di ambiti, come l'analisi di immagini, la segmentazione della clientela, i sistemi di raccomandazione e così via.

Una formulazione del clustering molto studiata in letteratura è quella del K-Center Clustering (che rientra anche nell'ambito dei problemi di Facility Location): dato come input un insieme di punti di uno spazio metrico e dato un parametro K, si cerca un sottoinsieme di K punti, chiamati *centri*, che minimizzi la massima distanza di un qualsiasi punto di input dal centro più vicino. I K centri inducono immediatamente una partizione dei punti in K cluster, dove ciascun punto è assegnato al cluster associato al centro più vicino. Tale problema è notoriamente NP-hard, e per tale motivo la letteratura si è concentrata sull'ottenimento in tempo polinomiale di soluzioni approssimate. In questo contesto, è fondamentale notare che gli algoritmi di clustering possono essere soggetti a bias: dato che essi hanno un impatto sempre più evidente sulla società, è importante sviluppare algoritmi in grado di tenere in considerazione criteri di fairness. Per tale motivo è stata sviluppata una variante del problema del K-Center Clustering chiamata Fair K-Center (FKC). Tale variante richiede, come vincolo aggiuntivo, che i centri trovati soddisfino un criterio di fairness: essi devono rappresentare un sottoinsieme indipendente di un matroide. Un matroide spesso utilizzato per rappresentare vincoli di fairness, e che sarà utilizzato anche in questa tesi, è il partition matroid, dove i punti sono suddivisi in categorie, e in un insieme indipendente per ogni categoria ci possono essere solo un certo numero k_i di punti. Un'ulteriore variante al problema introduce la possibilità di tenere in considerazione un certo numero di ou-

tliers, ovvero di punti che non sono considerati all'interno della soluzione finale (non vengono dunque messi in alcun cluster): questa variante è spesso riferita con il nome di Robust K-Center Clustering, o Robust Matroid se si utilizza un matroide, ma non verrà affrontata in questa tesi.

Per tali problemi sono stati proposti algoritmi efficienti per vari modelli: dapprima quello sequenziale, in cui si richiedeva di poter lavorare con l'intero dataset, cosa impensabile con l'aumentare della dimensione dello stesso. Inoltre, nella realtà spesso i dati arrivano in un flusso continuo, ed è dunque più conveniente pensare di occupare un po' di tempo all'arrivo di un nuovo punto piuttosto che aspettare che il dataset sia completo (soprattutto perchè spesso non sarà mai completo, dato che il flusso potenzialmente è infinito). Per questo motivo in Henzinger e al. [3] è stato introdotto il modello Streaming, e quindi algoritmi che, in tale modello, tengono in memoria solo un piccolo sottoinsieme dello stream stesso. Da esso è poi scaturito in Datar e Motwani [4] il modello Sliding Window, utilizzato in questa tesi, in cui vengono tenuti in considerazione in ogni istante solo gli N punti più recenti, dato che quelli più vecchi non sono considerati significativi (si pensi alla rilevazioni di usi sospetti di carte di credito, in cui è importante individuare anomalie nel pattern dei pagamenti recenti).

1.1 Contributo di questo lavoro

L'obiettivo della tesi è stato quello di implementare in modo ottimizzato l'algoritmo per il problema FKC nel modello Sliding Window proposto da Cappellotto e al. [1] e condurre un'analisi sperimentale accurata su dataset sia reali che sintetici, studiando le sue prestazioni al variare dei parametri essenziali che lo caratterizzano e confrontandole con l'attuale stato dell'arte, descritto in Chen e al. [2].

Il risultato sarà quello di dimostrare che l'algoritmo di Cappellotto supera molto spesso in termini di prestazioni (temporali e spaziali) quello di Chen, e al crescere della dimensione della finestra diventa l'unico praticabile, oltre ad essere molto facilmente adattabile ad esigenze prestazionali o di qualità.

1.2 Struttura del documento

Nel Capitolo 2 vengono presentate le nozioni di base per la comprensione e la presentazione degli algoritmi, con un breve inciso sui lavori finora svolti per i problemi di K-Center Clustering e Fair K-Center Clustering nei vari modelli sequenziale, Streaming e Sliding Window.

Nel Capitolo 3 viene presentato l'algoritmo di Cappellotto e al. [1], con le relative analisi di correttezza e complessità.

Nel Capitolo 4, dopo una breve premessa sull'ambiente di test vengono presentati i risultati sperimentali ottenuti confrontando l'algoritmo di Cappellotto e al. [1] e le sue derivazioni con l'algoritmo di Chen e al. [2] su vari dataset reali e sintetici. Gli esperimenti presentati sono solo un piccolo sottoinsieme significativo di quelli effettivamente eseguiti.

Nel Capitolo 5 vengono esposte le conclusioni del lavoro, con alcune possibili indicazioni per ulteriori sviluppi futuri.

Capitolo 2

Nozioni preliminari

Per iniziare formalizziamo alcuni concetti che saranno utili nella definizione del problema affrontato, e rivediamo i risultati della letteratura in merito al problema del K-Center Clustering e della sua variante FKC nei vari modelli.

2.1 Spazio metrico

Definizione 2.1: Spazio metrico *Uno spazio metrico è una struttura matematica costituita da una coppia* (W, d) *dove* W *è un insieme di punti (ground set), mentre* d *è una funzione di distanza su* W *che associa a due punti* x *e* y *di* W *un numero reale* $d(x, y)$ *tale che le seguenti proprietà sono vere* $\forall x, y, z \in W$*:*

- $d(x, y) \geq 0$*;*
- $d(x, y) > 0 <=> x \neq y$*;*
- $d(x, y) = d(y, x)$ *(simmetria);*
- $d(x, z) \leq d(x, y) + d(y, z)$ *(disuguaglianza triangolare).*

Un esempio importante, ma non l'unico, è lo spazio Euclideo $\mathbb{R}^D$. Nel seguito si assumerà che gli insiemi di punti utilizzati appartengano a uno spazio metrico.

2.1.1 Notazione di palla

Dato un insieme di punti W, un punto $x \in W$ e $r > 0$, definiamo la palla di centro x e raggio r, denotata con $B(x, r)$, come l'insieme di punti $y \in W$ tali che $d(x, y) \leq r$.

2.1.2 Doubling dimension

La doubling dimension è una caratteristica di uno spazio metrico (o di un sottoinsieme dei suoi punti) che verrà utilizzata solamente per l'analisi dell'algoritmo, e non per la sua descrizione.

Definizione 2.2 *Dato un insieme di punti W e un suo sottoinsieme W_t, la doubling dimension di tale sottoinsieme è il più piccolo numero D tale che ogni palla $B(x, r)$ con $x \in W_t$ è contenuta nell'unione di al massimo 2^D palle di raggio $\frac{r}{2}$ centrate in punti di W_t.*

Questo significa anche che una palla di raggio r può essere coperta, per $0 \leq \epsilon \leq 1$ da al massimo $2^{\lceil \log_2(1/\epsilon) \rceil}$ palle di raggio ϵr.

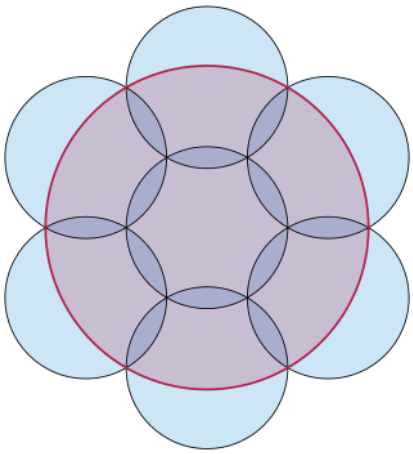

Figura 2.1: Nel piano euclideo, una palla di raggio r può essere coperta da 7 palle di raggio $\frac{r}{2}$, da cui la doubling dimension è $\log_2(7)$ (credits to [5]).

Il seguente risultato importante, la cui dimostrazione è presente in Dandolo e al. [6], verrà utilizzato nel seguito nell'analisi della complessità dell'algoritmo realizzato.

Lemma 2.3 *Dato un insieme di punti W di uno spazio metrico con doubling dimension D e un suo sottoinsieme $Y \subseteq W$ tale che ogni coppia di punti $a, b \in Y$ abbia distanza $d(a, b) > r$, allora per ogni $R \geq r$ e per ogni punto $x \in W$, si ha*

$$|B(x, R) \cap Y| \leq (\frac{4R}{r})^D$$

Questo significa che il numero di punti di Y contenuti in una palla di raggio $R \geq r$ è limitato da una funzione che dipende dalla doubling dimension.

2.2 Modello Sliding Window

L'algoritmo di Cappellotto e al. [1] è stato sviluppato per il modello Sliding Window, una variante del modello Streaming presentata in Datar-Motwani [4].

Nel modello Streaming standard, l'input viene fornito come un flusso continuo, possibilmente illimitato, di oggetti (nel nostro caso, punti), che arrivano uno alla volta.

Nel modello Sliding Window i punti arrivano nella stessa maniera, e al tempo t deve essere possibile ottenere una soluzione al problema prescelto per il sottoinsieme W_t che contiene gli ultimi N elementi arrivati nello stream, dove N, predeterminato e noto all'algoritmo, è la dimensione della finestra. Formalmente, definendo $t(p)$ il tempo di arrivo del punto p, in un generico istante t si ha che $W_t = \{p \,|\, 0 \leq t - t(p) < N\}$. Per convenienza, considereremo il tempo come suddiviso in step: ad ogni step un solo nuovo punto viene processato, ed eventualmente un solo altro punto espira, per mantenere la cardinalità della finestra $|W_t| \leq N$. Gli oggetti dello stream sono processati da un singolo processore, che ha a disposizione una quantità limitata di memoria che non può contenere tutti i punti di una finestra: per questo motivo l'obbiettivo è di ottenere una buona soluzione non utilizzando tutti i punti della finestra, ma solo un sottoinsieme.

In questi modelli di streaming le metriche di performance da ottimizzare sono:

- la quantità di memoria locale utilizzata;
- il tempo di update per ogni nuovo punto dello stream;
- il tempo di query, cioè di calcolo della soluzione in un certo istante di tempo.

2.2.1 Aspect ratio

Un importante caratteristica dello stream è il suo aspect ratio, che verrà utilizzato nell'algoritmo e nella sua analisi.

Definizione 2.4 *Dato uno stream di punti S di uno spazio metrico con una funzione distanza d, si definisce aspect ratio Δ di S il rapporto tra la massima e la minima distanza di due punti distinti all'interno dello stream:*

$$\Delta = \frac{maxDist}{minDist}$$

Similmente, ad ogni istante t, si definisce aspect ratio Δ_{W_t} della window corrente W_t il rapporto tra la massima e la minima distanza di due punti distinti all'interno di W_t:

$$\Delta_{W_t} = \frac{maxDist_{W_t}}{minDist_{W_t}}$$

2.3 Matroide

Il concetto di matroide è utilizzato per definire i vincoli aggiuntivi del problema in questione.

Definizione 2.5 *Dato W (ground set) un insieme di elementi di uno spazio metrico con distanza d, un matroide su tale insieme è definito come una coppia $M_W = (W, I_W)$ dove I_W è una famiglia di sottoinsiemi di W indipendenti per il matroide, che soddisfano le seguenti proprietà:*

- *l'insieme vuoto è indipendente ($\emptyset \in I_W$);*
- *ogni sottoinsieme di un insieme indipendente è a sua volta indipendente ($\forall X' \subseteq X : X \in I_W => X' \in I_W$);*
- *dati $A, B \in I_W$ con $|A| > |B|$, $\exists x \in A \backslash B$ tale che $B \cup \{x\} \in I_W$ (augmentation property).*

Un insieme indipendente è massimale se non è contenuto propriamente in un altro insieme indipendente. Una conseguenza dell'augmentation property è che ogni insieme indipendente massimale per un matroide ha la stessa cardinalità, che è definita come il rango del matroide $rank(M_W)$.

Un altro aspetto fondamentale per la realizzazione dell'algoritmo del Capitolo 3 è che, dato un matroide $M_W = (W, I_W)$ e un sottoinsieme $W' \subseteq W$, si può definire una restrizione del matroide su W' come $M_{W'} = (W', I_{W'})$ dove $I_{W'} = \{X \cap W' | X \in I_W\}$. Tale $M_{W'}$ è anch'esso un matroide.

Lemma 2.6 (Extended augmentation property) *Dato un matroide $M_W = (W, I_W)$, un suo insieme indipendente $A \in I_W$, un sottoinsieme $W' \subseteq W$ e un insieme indipendente $B \subseteq W'$ che sia massimale in W', se $\exists y \in W' \backslash A$ tale che $A \cup \{y\} \in I_W$, allora $\exists x \in B \backslash A$ tale che $A \cup \{x\} \in I_W$.*

2.3.1 Matroide di partizione

Il matroide utilizzato nel nostro lavoro è un matroide di partizione.

Definizione 2.7 *Data una partizione $W_1, W_2, ...W_\ell$ del ground set W (quindi insiemi $W_i \subseteq W$ tali che $\cup_{i=1}^{\ell} W_i = W$ e $\forall i, j$ tali che $1 \leq i < j \leq \ell : W_i \cap W_j = \emptyset$), e dati $k_1, k_2, ..., k_\ell$ interi positivi, la famiglia di insiemi indipendenti per il matroide di partizione è:*

$$I_W = \{X \subseteq W : \forall_{i=1}^{\ell} |X \cap W_i| \leq k_i\}$$

Questo significa che, per ogni insieme della partizione W_i, un insieme indipendente per il matroide di partizione deve contenere al più k_i elementi di tale insieme.

2.4 K-Center Clustering

Dato un insieme di punti W di uno spazio metrico con funzione distanza d, per un $p \in W$ e un $C \subseteq W$ indichiamo la minima distanza di tale punto p dal sottoinsieme C con:

$$d(p, C) = min_{q \in C} d(p, q)$$

e definiamo il raggio di C rispetto a W come la massima distanza di un punto di W da C:

$$r_C(W) = max_{p \in W} d(p, C).$$

Dato un intero positivo $K < |W|$, il problema del K-Center Clustering richiede di trovare un sottoinsieme $C \subseteq W$ di cardinalità K che minimizzi $r_C(W)$ (il raggio di tale soluzione). I punti di tale insieme C sono chiamati centri. Ogni soluzione ammissibile per tale problema induce naturalmente una suddivisione dei punti di W in cluster, che si ottiene assegnando ogni punto di W al suo centro più vicino.

Il raggio della soluzione ottima è:

$$OPT_{K,W} = min_{C \subseteq W, |C| \leq K} r_C(W)$$

2.4.1 Fair K-Center Clustering

Una variante del problema del K-Center Clustering è il problema del Fair K-Center Clustering: in esso, viene definito un matroide di partizione che suddivide i punti dello stream in ℓ gruppi, e si richiede che i centri restituiti come soluzione del problema siano un insieme indipendente per tale matroide. Questo significa che l'insieme dei centri deve includere al massimo k_i punti per ogni gruppo $1 \leq i \leq \ell$, con k_i tali che $\sum_{i=1}^{\ell} k_i = K$.

Nel modello Sliding Window si assume definito un matroide di partizione $M = (S, I_S)$ di rango K su tutto lo stream S (potenzialmente infinito), e ad un qualsiasi step t si deve poter restituire una soluzione al problema FKC che sia un insieme indipendente del matroide M_W ottenuto come proiezione di M sulla finestra W_t. Il raggio della soluzione ottima è:

$$OPT_{M_W} = min_{C \subseteq W, C \in I_S} r_C(W)$$

2.4.2 Altre varianti del problema

Al problema descritto esistono ulteriori varianti, che utilizzano altre matroidi (Matroid Center) o che permettono di avere un certo numero $0 \leq z \leq |W|$ di outliers (Robust K-Center), ovvero di escludere tali punti dal calcolo del raggio della soluzione. Per essere precisi, il problema del K-Center Clustering con z outliers vuole identificare un set T di K centri che minimizzi $max_{p \in W \setminus Z_T} d(p, T)$ dove Z_T è il set di z punti più distanti dai punti di T.

2.5 Problema di intersezione di matroidi (senza pesi)

Date due matroidi $M_1(V, I_1)$ e $M_2(V, I_2)$ definite sullo stesso ground set V, si vuole trovare un comune insieme indipendente $S \in I_1 \cap I_2$ di cardinalità massima.

Questo problema e la sua versione pesata (in cui ad ogni punto è associato un peso, e si vuole massimizzare il peso dell'insieme S) sono risolvibili in tempo lineare mediante la risoluzione di un matching su un grafo bipartito appositamente costruito, come ricordato in Schrijver e al. [7]. Esso è fondamentale per la risoluzione dell'algoritmo presentato da Chen e al. [2].

2.6 Lavoro correlato

Ricordando che il problema K-Center Clustering è in generale NP-hard (questo non è vero se i punti sono in una linea come dimostrato in Chen e al. [8], o in alcuni particolari spazi metrici particolari, in cui è risolvibile in tempo polinomiale), presentiamo ora alcuni importanti risultati ottenuti nei vari modelli possibili.

2.6.1 K-Center Clustering in modello sequenziale

In Gonzalez [9] e Hochbaum-Shmoys [10] viene proposto un algoritmo per la risoluzione approssimata del problema K-Center Clustering nel modello sequenziale: tale algoritmo genera una 2-approssimazione del raggio ottimo, ed esegue in tempo $O(K \cdot |W|)$. Inoltre, si dimostra che, se $P \neq NP$, una 2-approssimazione è il meglio ottenibile per il K-Center Clustering in tempo polinomiale.

Charikar [11] introduce il problema con outliers (Robust K-Center Clustering), e presenta un algoritmo per ottenere una 3-approssimazione nel modello sequenziale.

2.6.2 K-Center Clustering in modello Streaming

I primi lavori che presentano una $(2 + \epsilon)$-approssimazione al problema del K-Center Clustering nel modello Streaming sono McCutchen-Khuller [12] e Guha [13].

Anche in Ceccarello e al. [14] si analizza il problema del K-Center Clustering, con e senza outliers, nel modello Streaming: viene utilizzata, come in altri lavori (ad esempio in Malkomes e al. [15]), l'idea di tenere solo un sottoinsieme dell'input (coreset) per poi andare ad eseguire un algoritmo sequenziale su tale sottoinsieme. Questo permette una notevole flessibilità rispetto al risultato che si vuole ottenere: aumentando la dimensione del coreset aumenterà la precisione della soluzione trovata, mentre diminuendola diminuirà lo spazio richiesto per eseguire l'algoritmo.

In questo modo sono stati ottenuti un algoritmo che calcola una $(2+\epsilon)$-approssimazione per il K-Center Clustering nel modello MapReduce e Streaming, senza outliers (da notare che nel caso del modello Streaming, questo non migliora quanto già trovato in McCutchen-Khuller [12]) e un algoritmo che ritorna una $(3+\epsilon)$-approssimazione per il Robust K-Center Clustering nel modello MapReduce e Streaming (nel modello MapReduce, vengono proposti due algoritmi: uno deterministico e uno randomizzato che richiede meno memoria). Inoltre, implementando l'algoritmo con outliers nel modello sequenziale, si ottiene un grande miglioramento nel running time rispetto all'allora stato dell'arte di Charikar [11].

2.6.3 K-Center Clustering in modello Sliding Window

Per il modello Sliding Window il primo algoritmo è di Cohen-Addad [16], ed ottiene una $(6+\epsilon)$-approssimazione con memoria $O(K\epsilon^{-1}\log\Delta)$ dove Δ è l'aspect ratio dell'intero stream. Esso richiede tempo $O(K\epsilon^{-1}\log\Delta)$ per l'update della struttura e $O(K^2\epsilon^{-1}\log\Delta)$ per ritornare una soluzione al problema (query). Il problema fondamentale dell'algoritmo è la necessità di conoscere a priori l'aspect ratio dello stream. Nello stesso paper viene anche analizzato il problema della stima del diametro della window, ovvero la massima distanza tra due punti della finestra, che verrà utilizzato in seguito.

In Pellizzoni e al. [17] questo risultato viene migliorato, riuscendo a restituire una $(2+\epsilon)$-approssimazione usando una memoria $O(k\log(\Delta_W)(c/\epsilon)^{D_W})$ con $c>1$ costante e Δ_W e D_W rispettivamente aspect ratio e doubling dimension della window. Il tempo necessario è $O(M+K^2)$ per la procedura di update, e $O(K^2(\log(\log(\Delta_W))+(c/\epsilon)^{D_W}))$ per la query. Inoltre, viene introdotta l'idea di creare degli algoritmi indipendenti dall'aspect ratio, ovvero che non devono conoscere la massima e la minima distanza all'interno della window, sfruttando algoritmi di stima del diametro della finestra. L'algoritmo utilizzato nel seguito deve molto al lavoro di Pellizzoni e al. [17] per l'utilizzo sia dei validation che dei coreset points e per l'introduzione di un metodo per rendere tali algoritmi oblivious rispetto all'aspect ratio.

Le varianti con outliers sono analizzate in Pellizzoni e al. [18] e deBerg [19].

2.6.4 Fair K-Center Clustering nel modello sequenziale

In Chen e al. [2] viene dimostrato che il problema del K-Center Clustering con un qualsiasi matroide (e quindi anche il Fair K-Center Clustering) è NP-hard se si vuole approssimare con un fattore sotto al 2 (anche su una linea), e viene presentata una 3-approssimazione al problema stesso che ha tempo di esecuzione $\Omega(n^2 \log n)$ (oltre a una 7-approssimazione al problema con outliers, che generalizza il risultato di Charikar [11]). L'algoritmo senza outliers, utilizzato anche in Ceccarello e al. [20], verrà in seguito utilizzato all'interno del nostro algoritmo e come stato dell'arte del problema affrontato nel modello Sliding Window.

Anche in Chakrabarty-Negahbani [21] viene presentata una 3-approssimazione per il Fair K-Center Clustering nel modello sequenziale, ma questa è di più difficile implementazione, quindi verrà considerata solo la versione di Chen e al. [2].

2.6.5 Fair K-Center Clustering nel modello Streaming

In Chiplunkar [22] viene fornito un algoritmo $(3 + \epsilon)$-approssimato per il problema del Fair K-Center Clustering nel modello Streaming.

In Ceccarello e al. [20] il problema del Fair K-Center Clustering viene affrontato nel modello Streaming e MapReduce. L'idea molto interessante è costruire un coreset a partire da un primo clustering dei punti e selezionando un insieme indipendente da ciascun cluster, in modo da avere un intero set di alternative tra cui scegliere per la creazione dell'insieme di centri finale.

2.6.6 Fair K-Center Clustering nel modello Sliding Window

Per quanto riguarda il Fair K-Center Clustering nel modello Sliding Window, l'unico lavoro a riguardo è di Cappellotto e al. [1], il cui algoritmo verrà descritto e utilizzato nel seguito, che riesce ad ottenere una $(3 + \epsilon)$-approssimazione al problema stesso.

Capitolo 3

Algoritmo per Fair K-center Clustering con matroide di partizione nel modello Sliding Window

In questo capitolo passeremo alla descrizione dell'algoritmo di Cappellotto [1] e alla sua analisi di correttezza e complessità. Ricordiamo che si assume definito un matroide di partizione $M = (S, I_S)$ di rango K sull'intero stream S, e di volta in volta si farà riferimento a matroidi ottenuti per proiezione di M su un sottoinsieme di S.

3.1 Algoritmo

L'algoritmo descritto in Cappellotto e al. [1] è ottenuto a partire dall'algoritmo di Pellizzoni et al. [17] per la soluzione del problema del K-Center Clustering nel modello Sliding Window, integrandolo con le tecniche sviluppate in Ceccarello et al. [20] per soddisfare anche il vincolo di matroide.

L'idea è quella di fare dei guess del raggio ottimo per il problema K-Center Clustering sulla window, che vengono prese dalla progressione geometrica

$$\Gamma = \{(1+\beta)^i : \lfloor \log_{(1+\beta)} minDist \rfloor \leq i \leq \lceil \log_{(1+\beta)} maxDist \rceil\}$$

per un parametro $\beta > 0$ imposto dall'esterno. Per ogni guess γ vengono tenute sei strutture di supporto che presentano numerose simmetrie:

- tre insiemi di validazione, usati per scegliere il migliore guess per il raggio ottimo:

- AV_γ è l'insieme dei punti di validazione di attrazione: questo insieme contiene punti a distanza maggiore di 2γ l'uno dall'altro, e quando γ è un guess valido, rappresenta una possibile soluzione di raggio $r \leq 2\gamma$
- RV_γ, t è l'insieme dei punti di rappresentanza $repV_\gamma(v)$, uno per ogni $v \in AV_\gamma$, che rappresenta il punto più recente a distanza $d(v, repV_\gamma(v)) \leq 2\gamma$
- OV_γ è l'insieme che contiene i vecchi punti di rappresentanza, ovvero quelli per cui il $v \in AV_\gamma$ è espirato o è stato eliminato da AV_γ;

- tre coreset, usati per raffinare la soluzione trovata:
  - A_γ è l'insieme dei punti di attrazione del coreset a distanza maggiore di $\frac{\delta\gamma}{2}$ l'uno dall'altro, dove δ è un parametro scelto in funzione di ϵ, come vedremo in seguito;
  - R_γ è l'insieme dei punti di rappresentanza $repC_\gamma(a)$. Per ogni $a \in A_\gamma$, $repC_\gamma(a)$ è un insieme di punti tali che $d(a, repC_\gamma(a)) \leq \frac{\delta\gamma}{2}$, ed è un insieme indipendente massimale rispetto al matroide. Chiameremo $repC^i_\gamma(a)$ il sottoinsieme dei punti di $repC_\gamma(a)$ di categoria i per $a \in A_\gamma$
  - O_γ è uguale al corrispettivo OV_γ, ovvero mentiene i rappresentanti il cui punto di attrazione è espirato o eliminato da A_γ.

Il parametro δ in Cappellotto e al. [1] è univocamente definito dai parametri β, già descritto, ed $\epsilon \in (0,1)$, che caratterizza la qualità della soluzione da ricercare: l'algoritmo, come verrà dimostrato più avanti, calcola infatti una $(3+\epsilon)$-approssimazione del raggio ottimo. La formula esatta è $\delta = \frac{\epsilon'}{1+\beta}$ con $\epsilon' = \frac{\epsilon}{1+2\alpha}$, dove α rappresenta il coefficiente di approssimazione dell'algoritmo sequenziale utilizzato per il Fair K-Center Clustering (utilizzando l'algoritmo di Chen e al. [2], si ha $\alpha = 3$). Nel seguito, vedremo come nella pratica sia molto più comodo definire δ direttamente, e derivare da esso ϵ per sapere quale approssimazione si andrà ad ottenere. Riassumendo, i parametri di raffinamento da cui dipende il nostro algoritmo sono β e ϵ o δ.

L'algoritmo, il cui pseudocodice (tratto da Cappellotto e al. [1]) è presentato di seguito, viene suddiviso in due procedure:

- UPDATE(p): viene invocata allo step t, quando il nuovo punto p entra all'interno della finestra e un altro punto espira;
- QUERY: se invocata al tempo t determina una soluzione $(3 + \epsilon)$-approssimata per il problema FKC su W_t, eseguendo un algoritmo sequenziale A per il problema FKC sul coreset $R_\gamma \cup O_\gamma$ ottenuto scegliendo opportunamente il guess γ

Algoritmo 1 UPDATE(p)

Let $p \in W_i$ and let $repC^i_\gamma(p) = repC_\gamma(p) \cap W_i$
Let $TTL(p) = t(p) - t + |W_t|$ (time left to live for point p)
for each $\gamma \in \Gamma$ do
    # First, remove every expired point from $AV_{\gamma,t}, A_{\gamma,t}, OV_{\gamma,t}, O_{\gamma,t}$
    for each expired $v \in AV_{\gamma,t}$ do
        $AV_{\gamma,t} = AV_{\gamma,t} \backslash \{v\}$
        $RV_{\gamma,t} = RV_{\gamma,t} \backslash \{repV_\gamma(v)\}$
        $OV_{\gamma,t} = OV_{\gamma,t} \cup \{repV_\gamma(v)\}$
    for each expired $a \in A_{\gamma,t}$ do
        $A_{\gamma,t} = A_{\gamma,t} \backslash \{a\}$
        $R_{\gamma,t} = R_{\gamma,t} \backslash repC_\gamma(a)$
        $O_{\gamma,t} = O_{\gamma,t} \cup repC_\gamma(a)$
    for each expired $q \in OV_{\gamma,t}$ do
        $OV_{\gamma,t} = OV_{\gamma,t} \backslash \{q\}$
    for each expired $o \in O_{\gamma,t}$ do
        $O_{\gamma,t} = O_{\gamma,t} \backslash \{o\}$

    # Then find every point of attraction near to the new point p
    $EV = \{v \in AV_{\gamma,t} | d(p,v) \leq 2\gamma\}$
    $E = \{a \in A_{\gamma,t} | d(p,a) \leq \frac{\delta\gamma}{2}\}$

    # Check EV (attraction validation points)
    if $EV == \emptyset$ then
        $AV_{\gamma,t} = AV_{\gamma,t} \cup \{p\}$
        $repV_\gamma(p) = p$
        $RV_{\gamma,t} = RV_{\gamma,t} \cup \{repV_\gamma(p)\}$
        if $|AV_{\gamma,t}| > K + 1$ then
            $v_{old} = \text{argmin}_{v \in AV_{\gamma,t}} TTL(v)$
            $AV_{\gamma,t} = AV_{\gamma,t} \backslash \{v_{old}\}$
            $RV_{\gamma,t} = RV_{\gamma,t} \backslash \{repV_\gamma(v_{old})\}$
            $OV_{\gamma,t} = OV_{\gamma,t} \cup \{repV_\gamma(v_{old})\}$
        if $|AV_{\gamma,t}| > K$ then
            $t_{min} = min_{v \in AV_{\gamma,t}} TTL(v)$
            for each $a \in A_{\gamma,t}$ do
                if $TTL(a) < t_{min}$ then
                    $A_{\gamma,t} = A_{\gamma,t} \backslash \{a\}$
                    $R_{\gamma,t} = R_{\gamma,t} \backslash repC_\gamma(a)$
                    $O_{\gamma,t} = O_{\gamma,t} \cup repC_\gamma(a)\}$
            for each $q \in OV_{\gamma,t}$ do
                if $TTL(q) < t_{min}$ then
                    $OV_{\gamma,t} = OV_{\gamma,t} \backslash \{q\}$
            for each $o \in O_{\gamma,t}$ do
                if $TTL(o) < t_{min}$ then

```
                    O_{γ,t} = O_{γ,t} \ {o}
    else
        for each v ∈ EV do
            set repV_γ(v) = p in RV_{γ,t}

    #Check E (attraction coreset points)
    if E == ∅ then
        A_{γ,t} = A_{γ,t} ∪ {p}
        repC_γ(p) = {p}
        R_{γ,t} = R_{γ,t} ∪ repC_γ(p)
    else
        a_add = argmin_{a∈E} |repC^i_γ(a)| #break ties arbitrarily
        set repC_γ(a_add) = repC_γ(a_add) ∪ {p} in R_{γ,t}
        if |repC^i_γ(a_add)| > k_i then
            o_rem = argmin_{o∈repC^i_γ(a_add)} TTL(o)
            remove o_rem from repC_γ(a_add) in R_{γ,t}
```

Algoritmo 2 QUERY(p)

```
for increasing values of γ ∈ Γ with |AV_{γ,t}| ≤ K do
    C = ∅
    for each q ∈ AV_{γ,t} ∪ RV_{γ,t} ∪ OV_{γ,t} do
        if (C == ∅) or d(q, C) > 2γ then
            C = C ∪ {q}
        if |C| > K then
            break and move to next guess
    if |C| ≤ K then
        return A(R_{γ,t} ∪ O_{γ,t}, k)
```

Implementazione e miglioramenti

Nel corso dell'implementazione sono state fatte delle ottimizzazioni sull'algoritmo proposto, brevemente riportate di seguito:

- ordinando i punti all'interno di tutti i set utilizzati in base al loro tempo di arrivo, in Algoritmo 1 diventa $O(1)$ accedere al più vecchio punto di un dato insieme;
- la ricerca tra i guess nell'Algoritmo 2 è diventata una ricerca binaria;
- nell'Algoritmo 2 non è necessario scandire anche RV_γ, dato che sono tutti punti a distanza minore o uguale a 2γ dai punti di AV_γ, i quali invece sono aggiunti tutti. Inoltre, basta controllare se $|C| > K$ solo ogni volta che si aggiunge un punto a C.

Nell'implementazione è stato necessario implementare anche l'algoritmo A: si è scelto l'algoritmo descritto in Chen e al. [2], in quanto molto facilmente implementabile, anch'esso ottimizzato utilizzando una ricerca binaria e non lineare. Per la sua implementazione è stato necessario ricorrere alla risoluzione di un problema di intersezione di matroidi senza pesi. Per ulteriori informazioni si rimanda al codice[1].

3.2 Analisi teorica dell'algoritmo

In questa sezione verrà analizzata la correttezza dell'algoritmo, riprendendo quanto dimostrato in Cappellotto e al. [1], e verrà analizzata la complessità spaziale e temporale dell'algoritmo stesso. Per brevità verranno dimostrate solo le proprietà più importanti, per le restanti si rimanda alla tesi di Cappellotto e al. [1].

Riprendendo la notazione di Cappellotto e al. [1], denoteremo qualsiasi insieme X al tempo t come X_t. Questo significa ad esempio che l'insieme AV_γ al tempo t verrà denotato come $AV_{\gamma,t}$.

3.2.1 Correttezza

Dimostriamo che l'algoritmo restituisce una $(3+\epsilon)$-approssimazione per il problema del Fair K-Center Clustering.

Lemma 3.1 *Dati* $v' = \text{argmin}_{v \in AV_{\gamma,t}} t(v)$ *e* $a' = \text{argmin}_{a \in A_{\gamma,t}} t(a)$ *per un guess* $\gamma \in \Gamma$*, abbiamo che*

- $\forall q \in W_t$ *con* $t(q) \geq t(v')$ *:* $AV_{\gamma,t(q)} \subseteq AV_{\gamma,t} \cup AV_{\gamma,t(v')-1}$
- $\forall q \in W_t$ *con* $t(q) \geq t(a')$ *:* $A_{\gamma,t(q)} \subseteq A_{\gamma,t} \cup A_{\gamma,t(a')-1}$

Dimostrazione Concentriamoci su $AV_{\gamma,t}$: il suo comportamento è equivalente a quello di una coda. Infatti, nello step t possiamo rimuovere solo il più vecchio punto di $AV_{\gamma,t}$ (dato che un solo punto entra in ogni momento) ed eventualmente aggiungere il punto appena entrato nella finestra, che sarà sicuramente più nuovo di quelli già presenti in $AV_{\gamma,t}$. Quindi, dato che l'insieme di punti di una coda al momento intermedio $t(v') - 1 < t(q) \leq t$ è un sottoinsieme dei punti presenti nella coda ai tempi $t(v') - 1$ e t, possiamo concludere che $AV_{\gamma,t(q)} \subseteq AV_{\gamma,t} \cup AV_{\gamma,t(v')-1}$. Un discorso del tutto analogo vale per il set $A_{\gamma,t}$.

[1] Codice esperimenti https://github.com/FraVisox/k_center_with_matroid

Lemma 3.2 *Si consideri uno step t. Dato $v' = \text{argmin}_{v \in AV_{\gamma,t}} t(v)$, se esiste ($AV_{\gamma,t} \neq \emptyset$), per ogni $\gamma \in \Gamma$ valgono i seguenti invarianti dopo l'esecuzione di* UPDATE(p) *allo step t:*

1. $\forall q \in W_t$ *con* $t(q) \geq t(v')$ *si ha che* $d(q, RV_{\gamma,t} \cup OV_{\gamma,t}) \leq 4\gamma$*;*
2. $\forall q \in W_t$ *con* $t(q) < t(v')$*, se* $|AV_{\gamma,t}| \leq K$ *si ha che* $d(q, RV_{\gamma,t} \cup OV_{\gamma,t}) \leq 4\gamma$.

La seguente dimostrazione, riportata in Cappellotto e al. [1], segue da quanto dimostrato in Pellizzoni e al. [17] e Cohen-Addad [16].

Dimostrazione La dimostrazione dipende dalla dimensione di $AV_{\gamma,t}$ al tempo t:

1. se $|AV_{\gamma,t}| = 0$, allora $W_t = \emptyset$, dato che il primo punto ad entrare nella finestra è sempre aggiunto ad $AV_{\gamma,t}$. Quindi le proprietà sono valide, dato che $\nexists q \in W_t$.
2. se $|AV_{\gamma,t}| \geq 1$, consideriamo $v' = \text{argmin}_{v \in AV_{\gamma,t}} t(v)$ il punto più vecchio in $AV_{\gamma,t}$ al tempo t. In questo caso abbiamo due possibilità:

   (a) $\forall q \in W_t$ con $t(q) \geq t(v')$ abbiamo $d(q, RV_{\gamma,t} \cup OV_{\gamma,t}) \leq 4\gamma$.
   Se per assurdo esistesse $q' \in W_t$ con $t(q') > t(v')$ tale che $d(q', RV_{\gamma,t} \cup OV_{\gamma,t}) > 4\gamma$ (il punto non può essere $q' = v'$, dato che per ogni punto in $AV_{\gamma,t}$ esiste un suo rappresentante $z' \in RV_{\gamma,t}$ a distanza $d(q', z') \leq 2\gamma$), allora avremmo:
   - $d(q', RV_{\gamma,t}) > 4\gamma$ e quindi $d(q', AV_{\gamma,t}) > 2\gamma$ allo step t;
   - $d(q', OV_{\gamma,t}) > 4\gamma$ allo step t: dato che i punti in $OV_{\gamma,t}$ possono solo essere rappresentanti degli ultimi $K + 1$ punti che sono usciti da $AV_{\gamma,t}$, e dunque dovevano essere presenti in $AV_{\gamma,t(v')-1}$, questo significa che $d(q', RV_{\gamma,t(v')-1}) > 4\gamma$ e $d(q', AV_{\gamma,t(v')-1}) > 2\gamma$.

   Dato che, dal Lemma 3.1, sappiamo che $AV_{\gamma,t(q')-1} \subseteq AV_{\gamma,t} \cup AV_{\gamma,t(v')-1}$, sappiamo che $\forall v \in AV_{\gamma,t(q')-1} : d(q', v) > 2\gamma$ al tempo $t(q')$. Questo significa che il punto q' avrebbe dovuto essere aggiunto in $AV_{\gamma,t(q')}$, e dato che $t(q') \geq t(v')$, sarebbe ancora in $AV_{\gamma,t}$ (ASSURDO).

   (b) $\forall q \in W_t$ con $t(q) < t(v')$, se $|AV\gamma| \leq K$ si ha che $d(q, RV_{\gamma,t} \cup OV_{\gamma,t}) \leq 4\gamma$.
   Assumiamo, per assurdo, che esista $q' \in W_t$ con $t(q') < t(v')$ con $d(q', RV_{\gamma,t} \cup OV_{\gamma,t}) > 4\gamma$. Questo significa che $d(q, OV_{\gamma,t}) > 4\gamma$, ma dato che $|AV_{\gamma,t}| \leq K$, $\nexists v \in AV_{\gamma,t(v')}$ con $t(v') > t(v) > t - N$ altrimenti sarebbe ancora in $AV_{\gamma,t}$ e quindi v' non sarebbe il più vecchio punto di $AV_{\gamma,t}$ della finestra W_t (nessun punto è stato eliminato da $AV_{\gamma,t}$ nella finestra W_t). Questo implica che tutti i punti in $OV_{\gamma,t}$ erano rappresentanti di punti in $AV_{\gamma,t(q')-1}$. Quindi $d(q', RV_{\gamma,t(q')-1}) > 4\gamma$ e quindi $d(q', AV_{\gamma,t(q')-1}) > 2\gamma$. Quindi il punto q' avrebbe dovuto essere aggiunto in $AV_{\gamma,t(q')}$ e dato che $|AV_{\gamma,t}| \leq K$, sarebbe ancora in $AV_{\gamma,t}$ al tempo t (ASSURDO).

Similmente può essere dimostrato il seguente lemma:

Lemma 3.3 *Si consideri uno step t. Dato $a' = \text{argmin}_{a \in A_{\gamma,t}} t(a)$, se esiste, per ogni $\gamma \in \Gamma$ valgono i seguenti invarianti dopo l'esecuzione di* $\text{UPDATE}(\text{p})$ *allo step t:*

1. *$\forall q \in W_t$ con $t(q) \geq t(a')$ si ha che $d(q, R_{\gamma,t} \cup O_{\gamma,t}) \leq \delta\gamma$;*
2. *$\forall q \in W_t$ con $t(q) < t(a')$, se $|AV\gamma| \leq K$ si ha che $d(q, R_{\gamma,t} \cup O_{\gamma,t}) \leq \delta\gamma$.*

In Ceccarello e al. [20] viene dimostrato un lemma (Lemma 3) simile al seguente, per cui la prova verrà omessa. Ricordiamo, ai fini della notazione, che $OPT_{M_{W_t}}$ è il raggio della soluzione ottima del problema FKC sull'insieme di punti W_t con matroide M_{W_t}.

Lemma 3.4 *Si consideri uno step t e un guess $\gamma \in \Gamma$. Dato un coreset $Q \subseteq W_t$ con una funzione di prossimità $p : W_t \to Q$ che soddisfi le seguenti condizioni:*

1. *$\forall q \in W_t, d(q, p(q)) \leq \delta\gamma$;*
2. *Per ogni insieme indipendente $X \in I_{W_t}$, esiste una mappatura iniettiva $\pi_X : X -> Q$ tale che*
   - *$\{\pi_X(i) : i \in X\} \subseteq Q$ è un insieme indipendente $(\in I_Q)$;*
   - *per ogni $i \in X, d(i, \pi_X(i)) \leq \delta\gamma$.*

Allora:

1. *(P1) Esiste una soluzione al problema del K-Center Clustering con matroide $M_Q = (Q, I_Q)$ di costo al massimo $OPT_{M_{W_t}} + 2\delta\gamma$;*
2. *(P2) Ogni soluzione al problema del K-Center Clustering con matroide M_Q di costo r_F è anche una soluzione del problema di K-Center Clustering sull'insieme di punti W_t e matroide M_{W_t} di costo al massimo $r_F + \delta\gamma$.*

Lemma 3.5 *Dato p un punto che entra in W_t al tempo t, esso entra sempre all'interno di $R_{\gamma,t}$ al tempo $t = t(p)$ per ogni $\gamma \in \Gamma$.*
La dimostrazione a tale lemma è banale, basti osservare che possono verificarsi due casi:

1. se $E_t == \emptyset$, il punto è aggiunto in $A_{\gamma,t}$ e diventa il suo stesso rappresentante
2. se $E_t \neq \emptyset$, esso verrà sicuramente aggiunto come rappresentante di qualche punto in $A_{\gamma,t}$, e certamente non ne verrà eliminato

Lemma 3.6 $\forall \gamma \in \Gamma$, *dato* $a \in S$ *(nello stream, anche quelli che non sono presenti all'interno della window corrente) un punto che è entrato in* $A_{\gamma,t}$ *in un dato momento, per* $t \geq t(a)$ *abbiamo:*

- *se* $a \in A_{\gamma,t}$ *allora* $repC_{\gamma,t}(a)$ *è un insieme indipendente di cardinalità massima per* $W_t(a) = \{x \in W_t : (d(a,x) \leq \frac{\delta\gamma}{2}) \ \& \ (t(a) \leq t(x))\}$
- *se* $a \notin A_{\gamma,t}$ *allora* $repC_{\gamma,t}(a)$ *è un insieme indipendente di cardinalità massima per* $W_t(a) = \{x \in W_t : (d(a,x) \leq \frac{\delta\gamma}{2}) \ \& \ (t(a) \leq t(x) \leq \bar{t})\}$ *dove* $\bar{t} \leq t$ *è il tempo in cui* a *è stato rimosso da* $A_{\gamma,t}$

Dimostrazione Il lemma viene dimostrato per induzione sul tempo t:

1. CASO BASE: $t = t(a)$.
   Se tale punto è entrato in $A_{\gamma,t}$, l'insieme $repC_{\gamma,t}(a) = \{a\}$, il quale è ovviamente l'unico punto a distanza $d(x,a) \leq \frac{\delta\gamma}{2}$ da a. Dato che $W_t(a) = repC_{\gamma,t}(a)$, esso è un insieme indipendente di cardinalità massima per $W_t(a)$

2. CASO INDUTTIVO 1: $t > t(a) \ \& \ a \in A_{\gamma,t}$.
   Per ipotesi induttiva, $repC_{\gamma,t-1}(a)$ è un insieme indipendente di cardinalità massima per $W_{t-1}(a)$. Dato che $a \in A_{\gamma,t}$, significa che il punto u che espira da W_t al tempo t non appartiene a $W_t(a)$ (altrimenti sarebbe espirato anche a, e quindi non apparterrebbe a $A_{\gamma,t}$).
   Ci sono due casi:

   - se il punto che entra in W_t al tempo t non appartiene a $W_t(a)$, né $W_t(a)$ né $repC_{\gamma,t}(a)$ sono cambiati rispetto al tempo $t-1$, quindi per ipotesi induttiva $repC_{\gamma,t}(a)$ è un insieme indipendente di cardinalità massima per $W_t(a)$;
   - se il punto che entra in W_t al tempo t appartiene a $W_t(a)$, allora $W_t(a) = W_{t-1}(a) \cup \{p\}$ e $repC_{\gamma,t}(a) = repC_{\gamma,t-1}(a) \cup \{p\}$. A questo punto, viene rimosso un punto da $repC_{\gamma,t}(a)$ solo se abbiamo più di k_i punti di quella categoria nell'insieme stesso. Dato che un insieme indipendente di cardinalità massima non può avere più di k_i punti di quella categoria, rimane che $repC_{\gamma,t}(a)$ è un insieme indipendente di cardinalità massima per $W_t(a)$.

3. CASO INDUTTIVO 2: $t > t(a) \ \& \ a \notin A_{\gamma,t}$.
   L'ipotesi induttiva è la stessa del caso induttivo 1: dato che $a \notin A_{\gamma,t}$, siamo sicuri che il nuovo punto p entrato al tempo t non appartenga a $W_t(a)$. Anche qui abbiamo due casi:

   - se il punto u che espira al tempo t non è in $repC_{\gamma,t-1}(a)$, abbiamo che $repC_{\gamma,t}(a) = repC_{\gamma,t-1}(a)$, e dato che non sono stati aggiunti punti in $W_t(a)$, ma solo eventual-

mente rimossi, siamo sicuri che l'insieme indipendente massimale per $W_{t-1}(a)$ sia anche un insieme indipendente massimale per $W_t(a)$.

- se il punto u che espira al tempo t è in $repC_{\gamma,t-1}(a)$, dato che teniamo sempre i punti più nuovi per ogni categoria, significa che $repC_{\gamma,t}(a) \cap W_i = W_t(a) \cap W_i$ (ovvero tutti i punti di quella categoria presenti in $W_t(a)$ sono presenti nell'insieme dei rappresentanti). Dato che uno di questi punti viene rimosso e nessun nuovo punto entra in $W_t(a)$, segue che il rango di $W_t(a)$ è pari al rango di $W_{t-1}(a)$ meno 1. Quindi, dato che rimuoviamo un solo punto da $repC_{\gamma,t}(a)$, esso continua ad essere un insieme indipendente di cardinalità massima per $W_t(a)$.

A questo punto, non ci resta che provare la correttezza del nostro algoritmo.

Teorema 3.7 *Dati $\epsilon \in (0,1)$ e $\beta > 0$ e supponendo che sia disponibile un algoritmo sequenziale A di α-approssimazione per il problema FKC, quando la procedura* QUERY(p) *viene eseguita al tempo t il coreset $T = R_{\gamma,t} \cup O_{\gamma,t}$ possiede le proprietà del Lemma 3.4 con $\delta = \frac{\epsilon'}{1+\beta}$ e $\epsilon' = \frac{\epsilon}{1+2\alpha}$. Quindi, la soluzione calcolata eseguendo A su T è una $(\alpha+\epsilon)$-approssimazione per la soluzione del problema FKC su W_t.*

Dimostrazione La dimostrazione avverrà in diversi step:

1. STEP 1 (presente anche in Ceccarello e al. [20]): dimostriamo che il coreset $T = R_{\gamma,t} \cup O_{\gamma,t}$ soddisfa le condizioni del Lemma 3.4. Questo coreset è ritornato solo se $|AV_{\gamma,t}| \leq K$ e $|C| \leq K$.
   Dato che $|AV_{\gamma,t}| \leq K$, sfruttando quanto dimostrato nel Lemma 3.3 possiamo dire che $\forall q \in W_t : d(q,T) \leq \delta\gamma$. Quindi la condizione (C1) del Lemma 3.4 è soddisfatta.
   Dato un insieme $X \in I_{W_t}$, dobbiamo ora generare una funzione di mapping iniettiva che va da tale insieme indipendente al nostro coreset $\Pi_X : X -> T$ tale che:

   - $\{\Pi_X(o) | o \in X\} \in I_T$ (cioè i punti mappati formano un insieme indipendente contenuto in I_{W_t} e sono un sottoinsieme di T);
   - $\forall o \in X : d(o, \Pi_X(o)) \leq \delta\gamma$ (cioè ogni punto di X e la sua mappatura devono essere a distanza minore o uguale a $\delta\gamma$, quindi equivalentemente devono essere a distanza $\leq \frac{\delta\gamma}{2}$ da un punto di attrazione comune).

   Per costruire tale funzione, andiamo avanti un elemento di X alla volta ($X = \{x_u | 1 \leq u \leq |X|\}$): supponendo di aver già definito una mappatura per i primi $h \geq 0$ elementi di X e assumiamo dunque che $Y(h) = \{\Pi_X(x_u) | 1 \leq u \leq h\} \cup \{x_u | h < u \leq |X|\}$ sia un insieme indipendente ($\in I_{W_t}$). Per i primi h punti abbiamo che $d(x_u, \Pi_X(x_u)) \leq \delta\gamma$.

Considerando che per il Lemma 3.5, quando il punto x_{h+1} è stato aggiunto alla finestra, esso è stato sicuramente aggiunto a $R_{\gamma,t(x_{h+1})}$. A questo punto possiamo avere due possibilità:

- se $x_{h+1} \in R_{\gamma,t} \cup O_{\gamma,t}$, allora semplicemente poniamo $\Pi_X(x_{h+1}) = x_{h+1}$, da cui $Y(h+1) = Y(h)$ e $d(x_{h+1}, \Pi_X(x_{h+1})) = 0$;
- se $x_{h+1} \notin R_{\gamma,t} \cup O_{\gamma,t}$, sicuramente tale punto è entrato in $repC_{\gamma,t(x_{h+1})}(a')$ per un qualche $a' \in A_{\gamma,t(x_{h+1})}$, possiamo trovare $\Pi_X(x_{h+1})$ utilizzando il Lemma 3.6 e il Lemma 2.6: dato che $\exists x_{h+1} \in W_t(a') \backslash (Y(h) \backslash \{x_{h+1}\})$ tale che $(Y(h) \backslash x_{h+1}) \cup x_{h+1} \in I_{W_t(a')}$ e dato che $repC_{\gamma,t}(a')$ è insieme massimale per $W_t(a')$, allora siamo sicuri che $\exists \Pi_X(x_{h+1}) \in repC_{\gamma,t}(a') \backslash (Y(h) \backslash \{x_{h+1}\})$ tale che $Y(h) \backslash \{x_{h+1}\} \cup \{\Pi_X(x_{h+1})\} \in I_{W_t(a')}$. Dato che x_{h+1} e $\Pi_X(x_{h+1})$ sono entrambi a distanza $\leq \frac{\delta\gamma}{2}$ da un punto comune a', essi sono a distanza $\leq \delta\gamma$ l'uno dall'altro.

Avendo costruito in questo modo la funzione di mapping, rispettando le condizioni del Lemma 3.4, valgono le proprietà del Lemma 3.4 per il coreset T;

2. STEP 2: Dato che, per il Lemma 3.4 la soluzione ottima su M_T ha costo al più $OPT_{M_{W_t}} + 2\delta\bar{\gamma}$ dove $\bar{\gamma}$ è il guess scelto dalla procedura QUERY(), ed è dunque $OPT_{k,W_t} < \bar{\gamma} \leq OPT_{K,W_t}(1+\beta) \leq OPT_{M_{W_t}}(1+\beta)$. Questo perchè QUERY() si ferma quando trova non più di K punti a distanza maggiore di 2γ, e questo avviene certamente quando il guess è maggiore del raggio ottimo OPT_{K,W_t}, e dato il modo in cui è costruita la progressione geometrica, il primo guess maggiore di OPT_{k,W_t} è al massimo $OPT_{K,W_t}(1+\beta)$. Dato che una soluzione al problema FKC è anche una soluzione al problema K-Center su W_t, abbiamo che $OPT_{K,W_t}(1+\beta) \leq OPT_{M_{W_t}}(1+\beta)$.
   Quindi, al caso peggiore la soluzione ottima su M_T è al più $OPT_{M_{W_t}} + 2\delta\bar{\gamma} \leq OPT_{M_{W_t}} + 2\delta OPT_{M_{W_t}}(1+\beta) = OPT_{M_{W_t}}(1+2\epsilon')$. Dunque, la soluzione F trovata dall'algoritmo A su T ha costo $r_F \leq \alpha(1+2\epsilon')OPT_{M_{W_t}}$. Dalla proprietà P2 del Lemma 3.4 si ha che la soluzione F è anche soluzione per il problema FKC sull'intera finestra W_t, di costo al più $r_F + \delta\bar{\gamma} \leq r_F + \frac{\epsilon}{1+2\alpha}OPT_{M_{W_t}} \leq (\alpha+\epsilon)OPT_{M_{W_t}}$.

Utilizzando dunque un algoritmo in grado di ottenere una 3-approssimazione per il problema FKC nel modello sequenziale come quello di Chen e al. [2] si può ottenere una $(3+\epsilon)$-approssimazione per il problema FKC nel modello sliding window.

3.2.2 Analisi spaziale

Passiamo ora all'analisi della complessità spaziale.

Teorema 3.8 *Ad ogni step t, l'insieme di set in memoria (i coreset e gli insiemi di validazione per tutti i guess) contengono*

$$O(K^2 \frac{\log(\Delta)}{\log(1+\beta)} (\frac{32}{\delta})^{D_{W_t}})$$

punti, con D_{W_t} doubling dimension della finestra W_t e Δ aspect ratio dello stream S.

Dimostrazione Dato che $|\Gamma| = O(\frac{\log(\Delta)}{\log(1+\beta)})$, è sufficiente dimostrare che per ogni γ l'insieme dei punti tenuti in memoria è $O(K^2(\frac{32}{\delta})^{D_{W_t}})$.
I punti negli insiemi di validazione sono $O(K)$, come dimostrato sia in Cappellotto e al. [1] che Pellizzoni e al. [17], e addirittura la dimensione di ogni insieme ($AV_{\gamma,t}, RV_{\gamma,t}, OV_{\gamma,t}$) è $\leq K+1$.
Per quanto riguarda i coreset, dal Lemma 3.2 sappiamo che $\forall a \in A_{\gamma,t} : d(a, RV_{\gamma,t} \cup OV_{\gamma,t}) \leq 4\gamma$, quindi i punti in $A_{\gamma,t}$ possono essere chiusi in al più $2(K+1)$ palle di raggio 4γ. Dal Lemma 2.3 sappiamo che in ognuna di queste palle ci possono essere al più $(\frac{32}{\delta})^{D_{W_t}}$, il che implica che $|A_{\gamma,t}| \leq 2(K+1)(\frac{32}{\delta})^{D_{W_t}}$.
Dato che per ogni punto di $A_{\gamma,t}$ sono presenti in $R_{\gamma,t}$ al più K punti, $|R_{\gamma,t}| \leq 2K(K+1)(\frac{32}{\delta})^{D_{W_t}}$.
Per quanto riguarda $O_{\gamma,t}$, si può fare un ragionamento simile a $OV_{\gamma,t}$, come presentato sia in Cappellotto e al. [1] che Pellizzoni e al. [17], che limiti $|O_{\gamma,t}| \leq 2K(K+1)(\frac{32}{\delta})^{D_{W_t}}$.

3.2.3 Analisi temporale

Procediamo ora con l'analisi temporale delle due procedure $\mathrm{UPDATE(p)}$ e $\mathrm{QUERY()}$.

Teorema 3.9 *L'esecuzione di* $\mathrm{UPDATE(p)}$ *allo step t richiede tempo tempo*

$$O(K^2 \frac{\log(\Delta)}{\log(1+\beta)} (\frac{32}{\delta})^{D_{W_t}})$$

mentre la procedura $\mathrm{QUERY()}$ *esegue in tempo*

$$O(K^2 \frac{\log(\Delta)}{\log(1+\beta)} + K^2(\frac{32}{\delta})^{D_{W_t}} + T_A(R_{\gamma,t} \cup O_{\gamma,t}))$$

Dimostrazione In $\mathrm{UPDATE(p)}$ il tempo maggiore è speso nello svuotamento, per ogni guess $\gamma \in \Gamma$ degli orfani presenti in $O_{\gamma,t}$ più vecchi del più vecchio punto di attrazione di $AV_{\gamma,t}$. Al caso peggiore si può svuotare tutto $O_{\gamma,t}$ che dal Teorema 3.8 è $O(K^2(\frac{32}{\delta})^{D_{W_t}})$.
Per quanto riguarda $\mathrm{QUERY()}$, la complessità è dominata da:

- la costruzione di $T = R_{\gamma,t} \cup O_{\gamma,t}$ ($O(K^2(\frac{32}{\delta})^{D_{W_t}})$);
- il tempo richiesto ad eseguire A su T;

- il calcolo di C, che calcola per ogni $q \in AV_{\gamma,t} \cup RV_{\gamma,t} \cup OV_{\gamma,t}$ la $d(q, C)$, che può essere completata in tempo $O(K^2)$ per ogni guess (in totale $O(K^2 \frac{\log(\Delta)}{\log(1+\beta)})$)

Capitolo 4

Analisi sperimentale

Questo Capitolo viene introdotto da una breve descrizione dell'ambiente di test per poi concentrarsi sull'analisi sperimentale dell'algoritmo presentato, condotta con diversi scopi:

- confrontare le prestazioni dell'algoritmo ideato in Cappellotto e al. [1] e delle sue varianti descritte di seguito rispetto allo stato dell'arte Chen e al. [2];
- misurare l'impatto dei parametri di precisione β e δ, del numero di centri K, della dimensionalità del dataset e della dimensione N della window sulla qualità e sulle prestazioni temporali e spaziali dell'algoritmo di Cappellotto e al. [1] e delle sue varianti.

Ambiente di test

Tutti i test sono stati eseguiti su un cluster formato da 8 macchine, ciascuna con 500GB di RAM e processori AMD EPYC 7282 con 32 core, dotate di sistema operativo Rocky Linux 8.10. Ogni algoritmo è stato eseguito su solamente una di queste macchine, non utilizzando nessuno strumento di parallelizzazione.

Il codice è stato scritto in Java 1.8, utilizzando la libreria JGraphT [23] per la costruzione del grafo necessario per l'algoritmo di Chen e al. e il relativo algoritmo Push Relabel per il calcolo del massimo flusso (ulteriori informazioni sull'implementazione all'interno del codice [1] e sulla teoria alla base in Schrijver e al. [7]).

I risultati riportati sono la media aritmetica di 200 esecuzioni consecutive avvenute dopo il riempimento totale della finestra.

I grafici rilevano il comportamento degli algoritmi per quanto riguarda:

- il tempo di esecuzione della procedura $\mathrm{UPDATE(p)}$;

[1] Codice esperimenti https://github.com/FraVisox/k_center_with_matroid

- il tempo di esecuzione della procedura QUERY();
- il numero di punti mantenuti in memoria: a tale scopo, se un punto è memorizzato in diverse strutture dell'algoritmo, come ad esempio AV_γ e RV_γ, viene contato più volte;
- la qualità della soluzione ottenuta: dato che il problema FKC è in generale NP-hard e dunque non risolvibile esattamente, per misurare tale qualità è stato valutato il rapporto (ratio) rispetto a quello che dovrebbe essere il migliore algoritmo eseguito con gli stessi parametri. Per dimensioni di window fino a 30 000 punti tale algoritmo è CHEN, mentre per valori superiori in cui CHEN non può essere eseguito è PELLCAPP o, se anch'esso non riesce a terminare, in successione PELLCAPPDELTA05 e PELLCAPPDELTA10 (definiti nella Sezione successiva). Talvolta è anche presente il raggio della soluzione trovata da ciascun algoritmo.

Algoritmi analizzati

Lo stato dell'arte per il problema FKC nel modello Sliding Window è l'algoritmo sequenziale CHEN di Chen e al. [2], che ha però lo svantaggio di eseguire la procedura QUERY() sull'intera finestra, la quale deve essere dunque mantenuta interamente in memoria. L'algoritmo di Cappellotto e al. [1], invece, costruisce un coreset su cui eseguire la procedura QUERY() di CHEN per cercare di risparmiare memoria e tempo. La versione presentata nel Capitolo 3, denominata CAPP, necessita di conoscere l'aspect ratio della finestra durante la sua esecuzione: come esposto in Pellizzoni e al. [17], questo algoritmo può essere reso oblivious rispetto all'aspect ratio (cioè inconsapevole del suo valore). Per fare ciò, è necessario utilizzare un algoritmo di stima del diametro, che può essere l'algoritmo presentato in Cohen-Addad [16] o quello presentato nello stesso Pellizzoni e al. [17]. Chiameremo COHCAPP la variante oblivious che utilizza l'algoritmo di stima del diametro di Cohen-Addad [16] e PELLCAPP quella che utilizza l'algoritmo di stima del diametro di Pellizzoni e al. [17].

Come anticipato, è stato molto conveniente impostare non il valore di $\epsilon \in (0,1)$, ma quello di δ per poi calcolare il fattore di approssimazione $\epsilon = \delta(1+\beta)(1+2\alpha)$: in questo modo si ha un maggior controllo sul numero dei punti in memoria e sul tempo di esecuzione, pur non degradando eccessivamente la qualità della soluzione trovata. Questi algoritmi sono denominati CAPPDELTAxx, dove xx è sostituito dal valore di δ moltiplicato per 10 (ad esempio, se $\delta = 2$, il nome è CAPPDELTA20). Per questi algoritmi sono state implementate anche le versioni oblivious COHCAPPDELTAxx e PELLCAPPDELTAxx, che utilizzano rispettivamente l'algoritmo di stima del diametro di Cohen-Addad [16] e di Pellizzoni e al. [17].

L'ultimo tipo di algoritmo sviluppato è quello che possiede solamente i tre insiemi di validazione AV_γ, RV_γ e OV_γ, e dunque non utilizza gli insiemi coreset A_γ, R_γ e O_γ. Questi algoritmi,

denominati CAPPVAL, PELLCAPPVAL e COHCAPPVAL (rispettivamente non oblivious, oblivious con algoritmo di stima del diametro di Pellizzoni e al. [17] e oblivious con algoritmo di stima del diametro di Cohen-Addad [16]), restituiscono la stessa soluzione degli algoritmi CAPPDELTA40, PELLCAPPDELTA40 e COHCAPPDELTA40, ovvero con $\delta = 4$.

Dove non ulteriormente specificato, gli algoritmi testati utilizzano i parametri:

- dimensione della window $N = 10000$;
- $\beta = 2$;
- $\epsilon = 0.9$, a cui corrisponde $\delta = 0.043$ per CAPP, COHCAPP e PELLCAPP;
- $K = 14$, con k_i distribuiti in base alla popolosità delle categorie.

Dataset

I dataset utilizzati sono alcuni tra i più usati per il test di problemi di K-Center Clustering e FKC (Ceccarello e al. [20] e Pellizzoni e al. [17]). Innanzitutto sono stati considerati tre dataset reali:

- **PHONES** [24]: contiene 13 062 475 punti che rappresentano i dati letti dai sensori di vari telefoni. Ogni punto possiede un tag dell'attività svolta (stand, sit, walk, bike, stairs up, stairs down, null), considerato come la categoria di tale punto, e 3 dimensioni che rappresentano la posizione del telefono: i punti appartengono quindi allo spazio Euclideo $\mathbb{R}^3$;
- **HIGGS** [25]: contiene 11 milioni di punti che rappresentano le caratteristiche di particelle ad alta energia, generati tramite simulazioni Monte-Carlo. Ogni punto possiede un primo attributo binario che è utilizzato per stabilire la sua categoria e altre 28 caratteristiche, le cui ultime 7 sono funzione delle altre: sono stati considerati solo questi ultimi 7 attributi, quindi i punti appartengono allo spazio Euclideo $\mathbb{R}^7$;
- **COVERTYPE** (o **COVTYPE**) [26]: contiene 581 012 punti ottenuti da osservazioni geologiche di biomi forestali degli Stati Uniti. I punti possiedono 54 caratteristiche, di cui 44 binarie, e un attributo finale che ne indica la categoria (in totale sono presenti 7 categorie), per cui sono interpretati come punti dello spazio Euclideo $\mathbb{R}^{54}$.

Ad essi si aggiungono altri due dataset a più alta dimensionalità:

- **RANDOM**: è un dataset sintetico di punti dello spazio Euclideo $\mathbb{R}^{20}$ appartenenti a 6 categorie, di cui ogni coordinata è un numero randomico compreso tra 0 e 1. Dato il modo in cui sono stati generati, non è detto che esistano dei cluster all'interno di questo dataset;

- **NORMALIZED COVERTYPE** (o **NORMALIZED**): è il dataset COVERTYPE in cui ogni coordinata è normalizzata in modo da essere un numero compreso tra 0 e 1. Infatti, delle 54 coordinate di COVERTYPE solo 10 sono non binarie: le coordinate binarie non hanno molto peso, per cui gli algoritmi si comportano con il dataset COVERTYPE come se esso avesse una dimensionalità minore.

Nelle tabelle 4.1 e 4.2 è presente un riassunto dei parametri di tali dataset: dimensionalità, numero di punti, distanze massime e minime, numero di categorie e percentuale di punti per categoria.

Dataset	Dimensionalità	Numero di punti	Distanza massima	Distanza minima
PHONES	3	13 062 475	52.6	$8.1 \cdot 10^{-5}$
HIGGS	7	11 000 000	46.9	0.002
COVERTYPE	54	581 012	8853.4	2.82
RANDOM	20	1 000 000	3.33	0.33
NORMALIZED	54	581 012	2.89	0.00058

Tabella 4.1: Caratteristiche generali dei punti dei dataset utilizzati nell'analisi. Per il dataset PHONES sono riportate le distanze relative ai primi 600 000 punti, mentre per gli altri le distanze sull'intero dataset.

Nei test successivi sono stati utilizzati i valori di massima e minima distanza solo sui primi 600 000 punti (o meno, nel caso di COVERTYPE e NORMALIZED, che ne hanno solo 581 012) invece che quelli sull'intero stream o sulla dimensione dello stream fino a quel momento: questo non ha comportato nessuna significativa differenza, come dimostrato da alcuni esperimenti omessi per brevità. Per quanto riguarda le categorie, invece, è stato fissato un valore di K e da lì si sono trovati i valori k_i in funzione della popolosità della categoria. I valori di distanza e k_i utilizzati si trovano nel codice [2]

Dataset	Numero categorie	Distribuzione categorie
PHONES	7	**0**: 14.1%, **1**: 15.2%, **2**: 14.2%, **3**: 16.8%, **4**: 13.6%, **5**: 12.4%, **6**: 13.7%
HIGGS	2	**0**: 47%, **1**: 53%
COVERTYPE	7	**0**: 36.5%, **1**: 48.8%, **2**: 6.2%, **3**: 0.5%, **4**: 1.6%, **5**: 3%, **6**: 3.4%
RANDOM	6	**0**: 16.6%, **1**: 16.7%, **2**: 16.6%, **3**: 16.7%, **4**: 16.7%, **5**: 16.7%
NORMALIZED	7	**0**: 36.5%, **1**: 48.8%, **2**: 6.2%, **3**: 0.5%, **4**: 1.6%, **5**: 3%, **6**: 3.4%

Tabella 4.2: Caratteristiche delle categorie dei dataset utilizzati nell'analisi

[2]Codice esperimenti: https://github.com/FraVisox/k_center_with_matroid

Gli esperimenti sulla variazione della dimensionalità presenti nella Sezione 4.2.5 utilizzano altri dataset sintetici, chiamati BLOBS: essi sono stati ottenuti con la funzione $\mathrm{make\_blobs}$ della libreria Sci-Kit Learn [3]. Questi dataset possiedono 1 000 000 punti dello spazio Euclideo $\mathbb{R}^D$, dove $D \in \{5, 10, 15, 20, 25, 30, 35, 40, 45, 50\}$ è la dimensionalità del dataset, hanno 7 categorie e sono organizzati in cluster attorno a 21 centri generati all'interno del cubo $(0{,}100)^D$ con deviazione standard attorno ad essi di 2. Dato che si tratta di dataset sintetici, la distribuzione dei punti è omogenea per ogni categoria. I valori di massima e minima distanza sono presenti nella tabella 4.3, in cui si nota che all'aumentare della dimensionalità è presente una diminuzione dell'aspect ratio e un aumento delle distanze (quindi probabilmente aumenterà anche il raggio ottimo).

Dimensionalità	Distanza massima	Distanza minima	Aspect Ratio
5	163.8	0.04	4 095
10	210	0.65	323
15	235.4	1.6	147
20	277.8	2.66	104
25	278.5	3.36	83
30	323.9	4.36	74
35	334.4	5.83	57
40	341.8	6.85	50
45	366.1	7.4	49
50	365.5	8.76	42

Tabella 4.3: Caratteristiche generali dei punti dei dataset BLOBS utilizzati nell'analisi della doubling dimension

4.1 Ottenimento di algoritmi rappresentativi

L'obiettivo della prima fase di esperimenti è quello di ottenere pochi algoritmi rappresentativi delle prestazioni generali di tutti gli altri e decidere su quali dataset eseguirli. La conclusione sarà di utilizzare nella fase successiva:

- per i dataset PHONES, COVERTYPE e NORMALIZED le versioni ottenute dallo scambio in maniera randomica dei punti, mentre i dataset RANDOM e HIGGS rimarranno invariati;
- solo gli algoritmi $\mathrm{PELLCAPP}$, $\mathrm{PELLCAPPDELTAxx}$ e $\mathrm{PELLCAPPVAL}$, in quanto molto simili alle varianti non oblivious o oblivious con algoritmo di stima del diametro di Cohen-Addad [16].

[3] Documentazione Sci-Kit Learn: https://scikit-learn.org/stable/api/sklearn.datasets.html

4.1.1 Confronto tra dataset originali e con punti randomizzati

Questo primo insieme di esperimenti ha lo scopo di stabilire se scambiare i punti dei dataset in maniera randomica possa essere utile per rendere più generalizzabili i risultati degli esperimenti eseguiti successivamente. Questa idea deriva dall'osservazione della distribuzione dei dati all'interno di PHONES: dato che in questo dataset i punti sono ordinati per vicinanza e categoria, utilizzarlo significa trattare un caso molto particolare del problema FKC. Un problema simile si poteva riscontrare anche con HIGGS, COVERTYPE e NORMALIZED, anche se meno accentuato, mentre RANDOM, in quanto sintetico, è stato creato appositamente in modo da non presentare disomogeneità.

Confrontando l'esecuzione degli algoritmi su queste due tipologie di dataset (originali e con punti scambiati), si nota che per HIGGS i valori rimangono pressochè identici (per questo motivo, nel seguito, esso rimarrà nella versione originale, in quanto già sufficientemente omogeneo), mentre le differenze più evidenti sono in PHONES: il raggio (figura 4.3) è notevolmente inferiore nel caso del dataset originale, così come il tempo di $\mathrm{QUERY}()$ (figura 4.2), nonostante il tempo di $\mathrm{UPDATE(p)}$ (figura 4.1) e la memoria utilizzata (figura 4.4) siano maggiori. Questo può essere facilmente spiegato osservando che, se il raggio ottimo è minore, sono presenti più guess validi, e dunque ci sarà un numero maggiore di guess che non ha limitazioni sul numero di punti presenti in OV_γ, O_γ e A_γ, da cui scaturirà anche un aumento del tempo di esecuzione della procedura $\mathrm{UPDATE(p)}$. Inoltre, avendo punti molto vicini, il coreset costruito sarà molto più piccolo, perchè ci saranno meno cluster, per cui il tempo di $\mathrm{QUERY}()$ sarà minore. Per COVERTYPE e NORMALIZED le differenze non sono così marcate, ma è stato scelto ugualmente di utilizzare il dataset randomizzato, in quanto più omogeneo.

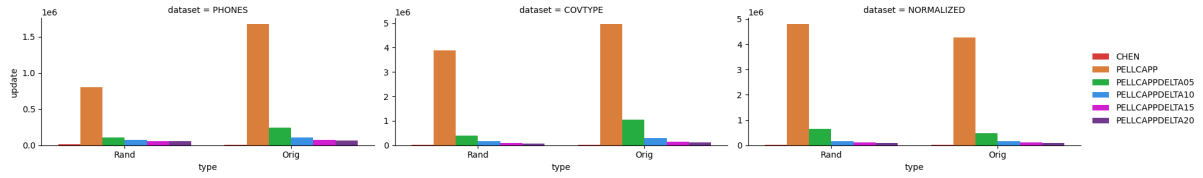

Figura 4.1: Tempo di esecuzione della procedura UPDATE(p) in dataset originali e con punti randomizzati

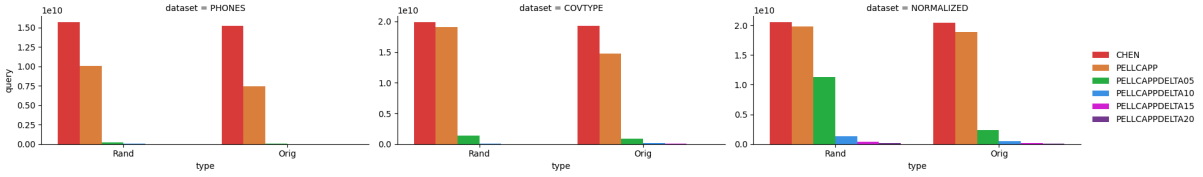

Figura 4.2: Tempo di esecuzione della procedura QUERY() in dataset originali e con punti randomizzati

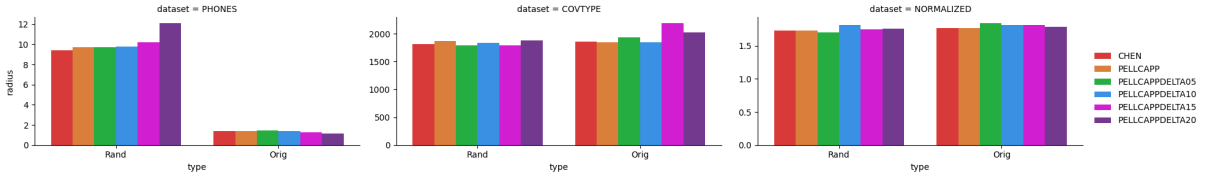

Figura 4.3: Raggio della soluzione trovata in dataset originali e con punti randomizzati

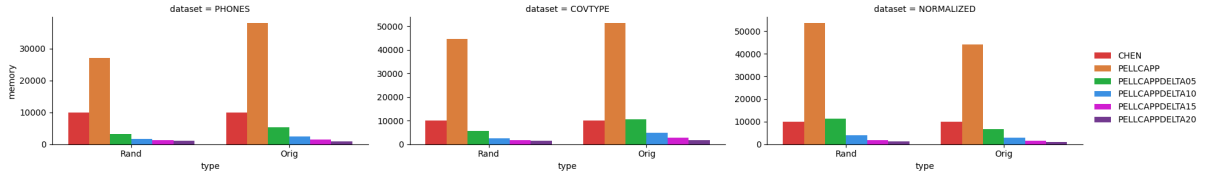

Figura 4.4: Numero di punti mantenuti in memoria in dataset originali e con punti randomizzati

4.1.2 Confronto tra algoritmi che utilizzano l'aspect ratio con le corrispondenti versioni oblivious

Stabilito l'utilizzo della versione randomizzata per tutti i dataset tranne HIGGS e RANDOM, passiamo ora a confrontare gli algoritmi che conoscono e utilizzano l'aspect ratio dello stream (le versioni CAPP, CAPPDELTAxx e CAPPVAL) con quelli oblivious allo stesso (le versioni COHCAPP, PELLCAPP, COHCAPPDELTAxx, PELLCAPPDELTAxx, COHCAPPVAL e PELLCAPPVAL). Le prime versioni hanno il grande svantaggio di richiedere di conoscere a priori l'aspect ratio dello stream fino a quel momento: questo in generale aumenta notevolmente la complessità di implementazione, perchè in ogni istante devono essere calcolati la distanza minima e massima. Per l'esecuzione dei nostri esperimenti sono state calcolate le distanze massime e minime sui primi 600 000 punti, per poi utilizzare tali valori al posto dei valori esatti da calcolare in ogni istante (questo procedimento è stato verificato tramite ulteriori esperimenti, omessi per brevità, che hanno dimostrato che utilizzare l'aspect ratio dei primi 600 000 punti non è molto differente da usare quello esatto dello stream fino a quel momento).

Questo insieme di esperimenti, i cui risultati sono riportati nelle figure 4.5, 4.6, 4.7 e 4.8, dimostra che per dataset a bassa dimensionalità non ci sono significative differenze tra gli algoritmi oblivious all'aspect ratio e quelli che non lo sono. Per i dataset ad alta dimensionalità (RANDOM e NORMALIZED), invece, conoscendo l'aspect ratio è più vantaggioso utilizzare la versione CAPP rispetto a COHCAPP e PELLCAPP: il numero di punti in memoria mantenuti da questi ultimi è circa $\frac{4}{3}$ rispetto alla corrispondente versione non oblivious. Questo vantaggio non è però presente nelle versioni CAPPDELTAxx e CAPPVAL.

Si può dunque concludere che gli algoritmi di stima del diametro della finestra sono molto accurati con δ sufficientemente grande, e che non sono presenti significative differenze nel-

l'utilizzo dell'algoritmo di Cohen-Addad [16] e Pellizzoni e al. [17]. Per questo motivo, nei test successivi non verranno prese in considerazione le versioni COHCAPP e CAPP, sebbene quest'ultima sia in generale da preferire su dataset ad alta dimensionalità se si conosce l'aspect ratio anche su un numero maggiore di punti, ma solamente la versione PELLCAPP (e dunque PELLCAPPDELTAxx e PELLCAPPVAL).

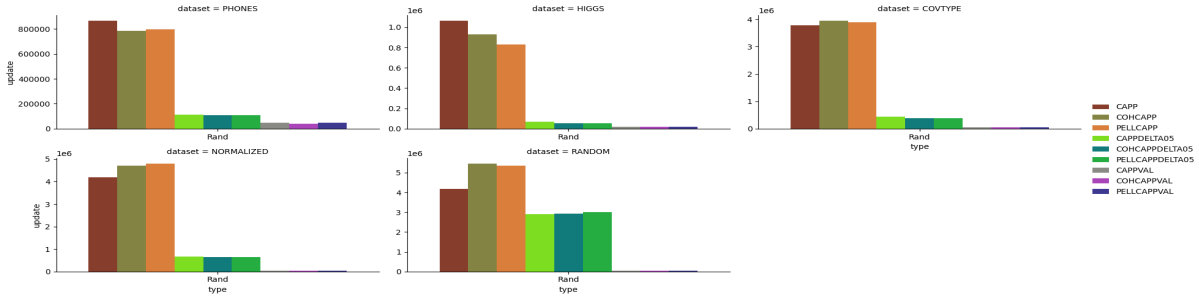

Figura 4.5: Tempo di esecuzione della procedura UPDATE(p) di algoritmi oblivious e non

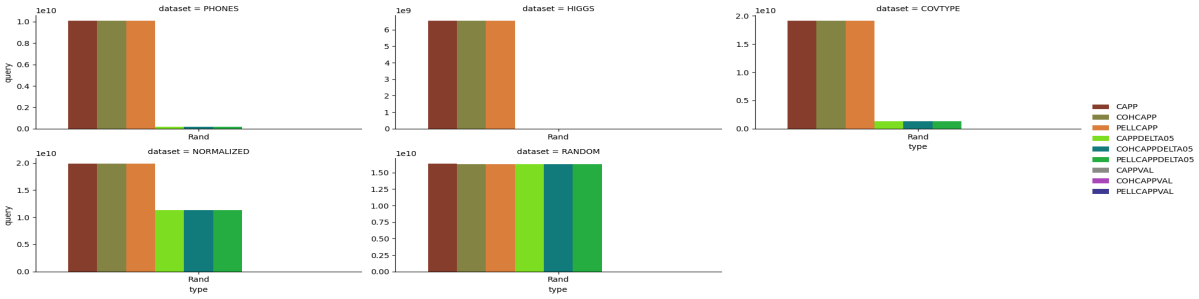

Figura 4.6: Tempo di esecuzione della procedura QUERY() di algoritmi oblivious e non

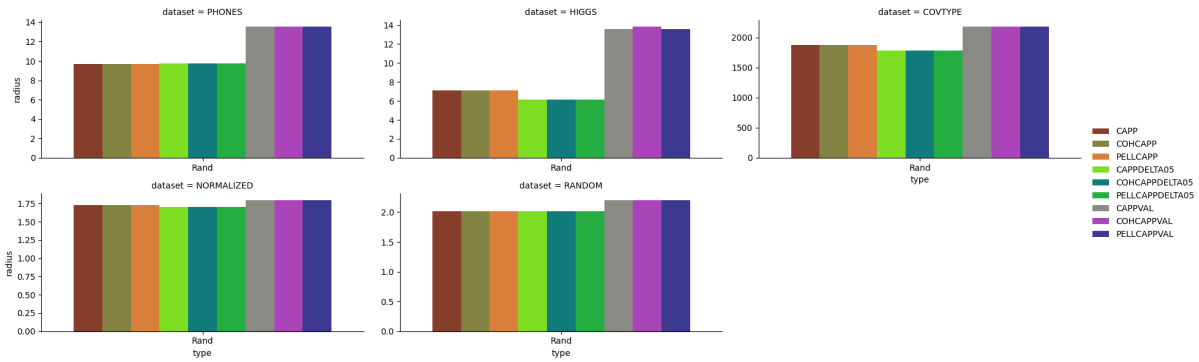

Figura 4.7: Raggio della soluzione trovata di algoritmi oblivious e non

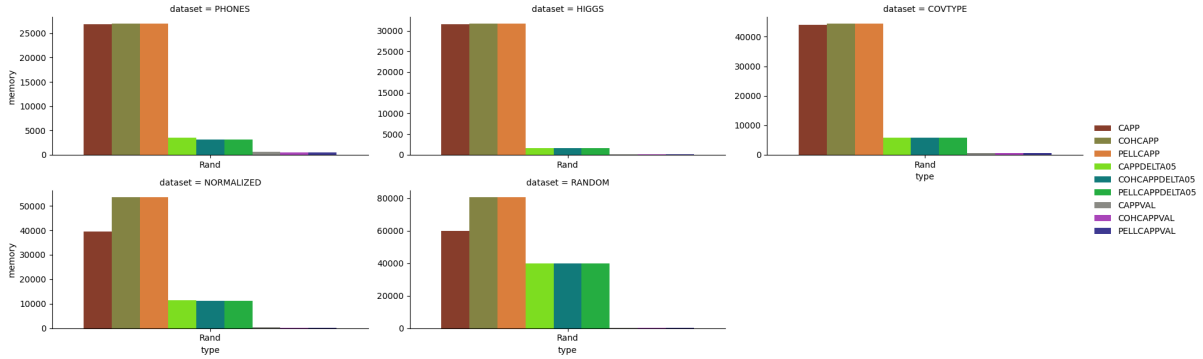

Figura 4.8: Numero di punti mantenuti in memoria di algoritmi oblivious e non

4.1.3 Confronto con versioni che cercano di restituire sempre K centri

Un problema che inizialmente sembrava significativo è il fatto che gli algoritmi (CHEN in primo luogo, ma anche CAPP e varianti) spesso ritornano meno di K centri: per questo motivo sono stati ideati una serie di algoritmi, chiamati Kxxxx, dove xxxx è il nome dell'algoritmo da cui derivano (ad esempio KPELLCAPPDELTA20 deriva da PELLCAPPDELTA20) che calcolano sempre un risultato con il massimo numero di centri (questo può non essere K se non ci sono abbastanza punti per tutte le categorie). In essi viene eseguito l'algoritmo originale da cui derivano, per poi aggiungere all'insieme dei centri C della soluzione trovata i punti mantenuti in memoria più distanti da C, sempre tenendo conto che C deve essere un insieme indipendente per il matroide di partizione considerato ($C \in I_W$). Questi algoritmi aggiungono ovviamente un overhead temporale, e negli esperimenti, omessi per brevità, risulta che il raggio calcolato è sempre uguale al raggio della soluzione trovata dagli algoritmi che non sfruttano questa strategia. Per questo motivo, gli algoritmi Kxxxx non verranno ulteriormente presi in considerazione.

4.2 Variazioni parametri

Passiamo ora alla fase di valutazione del comportamento degli algoritmi al variare dei parametri impostati. La conclusione di questa seconda fase sarà che gli algoritmi PELLCAPPDELTAxx sono in generale da preferire sia rispetto a CHEN, in quanto hanno tempi di esecuzione drasticamente minori e riescono ad eseguire anche su finestre molto grandi, che rispetto a PELLCAPP, data la maggior semplicità di definizione della precisione voluta e della memoria da occupare.

4.2.1 Variazione di δ

Il valore di δ è un parametro di precisione, che asserisce quanto grandi (e quindi precisi) devono essere gli insiemi coreset. Dato che il valore di $\epsilon \in (0,1)$, valido solo per PELLCAPP, genera un valore di δ molto basso, al di sotto di 0.047, osserviamo come cambia il comportamento di PELLCAPPDELTAxx per valori più alti di δ, ovvero $\delta \in \{0.5, 1.0, 1.5, 2.0, 2.5, 3.0, 3.5, 4.0\}$. E' importante notare che imporre $\delta = 4$ equivale a non utilizzare gli insiemi coreset, quindi la soluzione ottenuta in quel caso è pari a quella che si otterrebbe da PELLCAPPVAL: per questo motivo PELLCAPPVAL non verrà ulteriormente preso in considerazione.

Dai test eseguiti emerge che il cambiamento principale al crescere di δ è quello di una diminuzione del numero di punti in memoria (figura 4.12): per diminuire la precisione, dato che δ è direttamente proporzionale a ϵ, ogni guess preserverà meno punti negli insiemi coreset. Da ciò deriva in primo luogo una dimensione dei vari insiemi di gran lunga minore, e quindi un diminuito tempo di UPDATE(p) (figura 4.9), ma soprattutto una notevole diminuzione della dimensione del coreset $T = R_\gamma \cup O_\gamma$, su cui sarà dunque più vantaggioso eseguire CHEN. Infatti, il tempo di QUERY() subisce una drastica diminuzione, rendendo dunque possibile eseguire tale algoritmo anche su finestre di grande ampiezza (la scala usata nella figura 4.10 è logaritmica: tra $\delta = 0.5$ e $\delta = 1.5$ si nota una diminuzione di almeno un ordine di grandezza in tutti i dataset). La diminuzione della precisione, per quanto teoricamente possa sembrare eccessiva (con $\delta = 1$ il valore di ϵ è 21), nella pratica risulta essere accettabile: si rimane sempre all'interno di un fattore 2 rispetto al migliore algoritmo possibile (figura 4.11). E' anche curioso osservare che in generale non è detto che all'aumentare di δ il raggio aumenti, anche se ciò dipende molto dal singolo dataset.

Per semplificare la trattazione successiva, in seguito verranno comparati solo gli algoritmi con $\delta \in \{0.5, 1.0, 1.5, 2.0\}$, pur sapendo che aumentando δ sarebbe possibile ottenere prestazioni temporali e spaziali migliori, diminuendo in parte la precisione della soluzione.

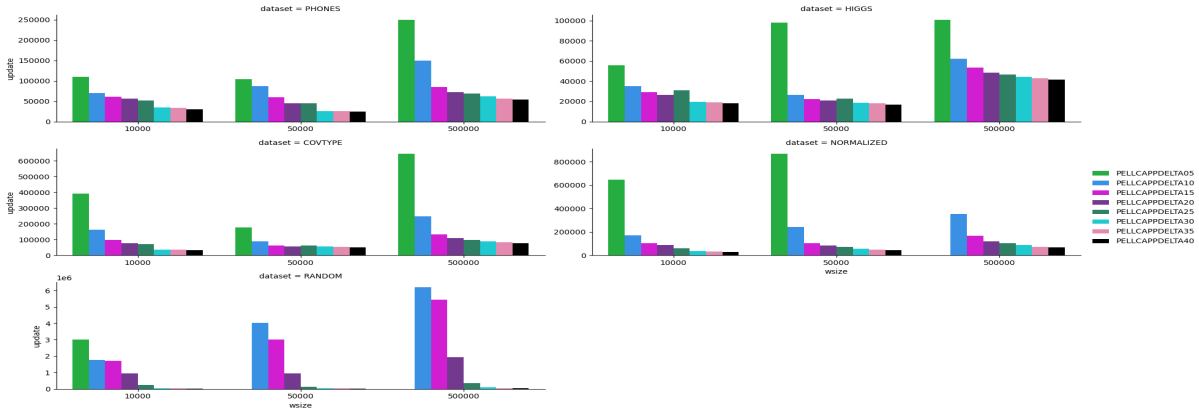

Figura 4.9: Tempo di esecuzione della procedura UPDATE(p) al variare di δ.

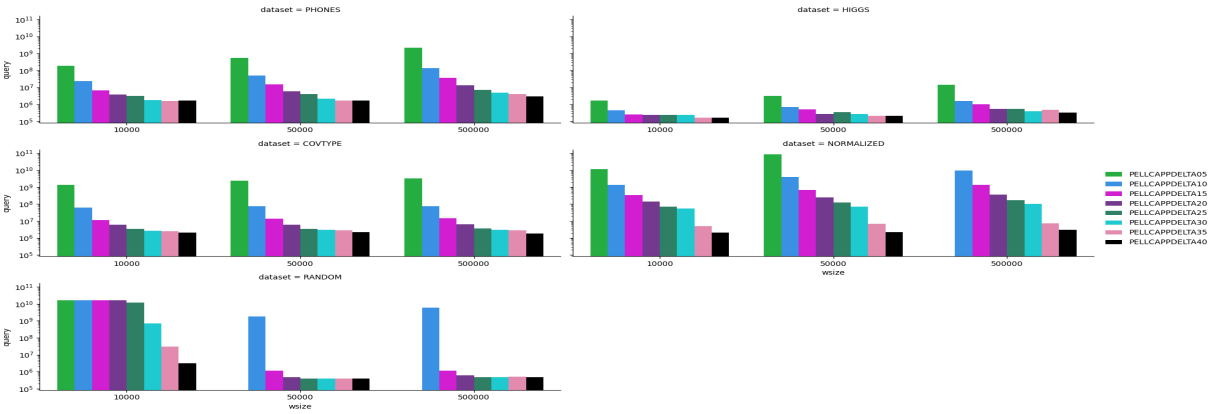

Figura 4.10: Tempo di esecuzione della procedura QUERY() al variare di δ. Questo è l'unico caso in cui la scala usata è logaritmica.

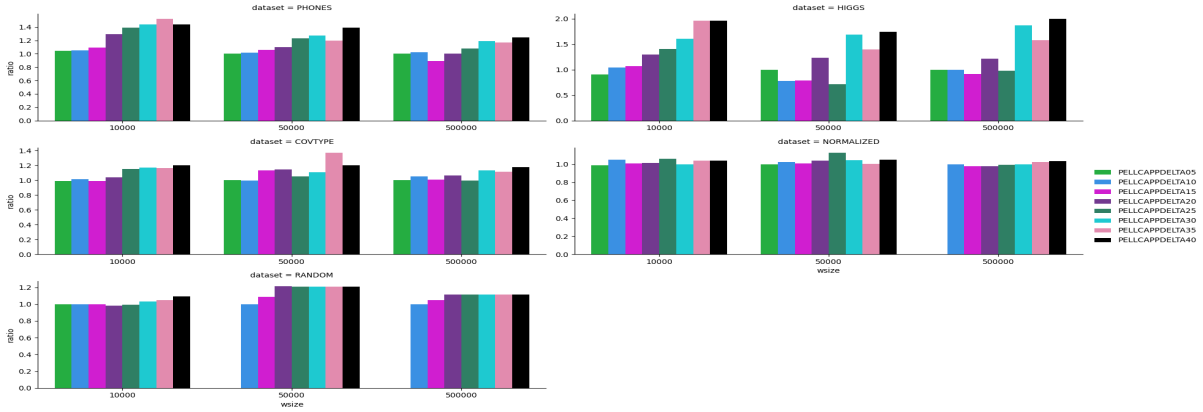

Figura 4.11: Ratio rispetto al migliore algoritmo (CHEN, PELLCAPP o PELLCAPPDELTAxx con δ minore possibile) al variare di δ.

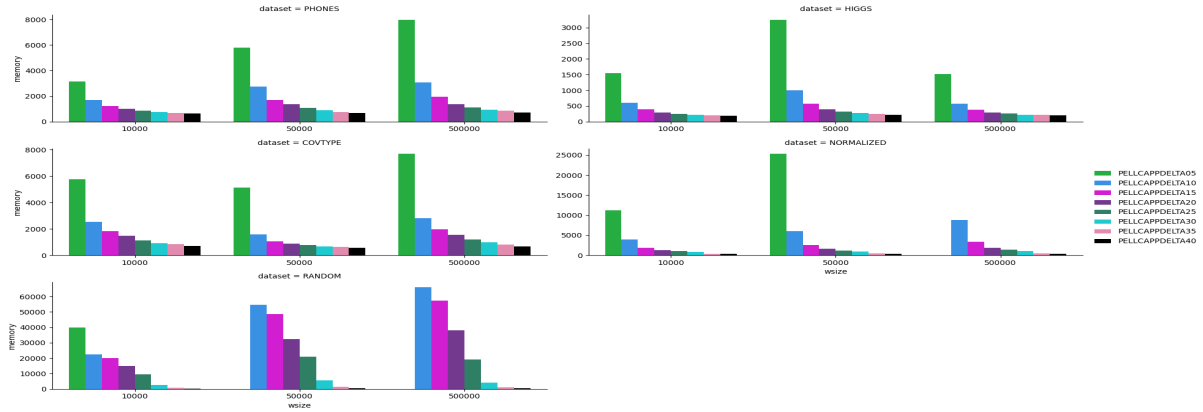

Figura 4.12: Numero di punti mantenuti in memoria al variare di δ

4.2.2 Variazione della dimensione della finestra

Il più grande vantaggio dell'algoritmo PELLCAPPDELTAxx rispetto sia a CHEN che a PELLCAPP è visibile nel loro comportamento al variare della dimensione N della finestra: in primo luogo, CHEN e PELLCAPP non possono venire eseguiti per dimensioni maggiori di 30 000 o 40 000 punti a seconda del dataset, e anche per dimensioni in cui riescono ad essere eseguiti presentano una crescita esponenziale per il tempo di QUERY() (figura 4.14) e lineare per il numero di punti in memoria (figura 4.16). PELLCAPP riesce, tenendo un numero di punti maggiore di CHEN, ad avere un tempo di QUERY() minore, ma il tempo di UPDATE(p) è di molto superiore (figura 4.13). Gli algoritmi del tipo PELLCAPPDELTAxx riescono invece ad eseguire fino (e probabilmente oltre) a finestre di dimensione di 500 000 punti, e il loro vantaggio principale è quello di tenere meno punti in memoria di CHEN: questo significa innanzitutto avere un tempo di UPDATE(p) assimilabile a quello di CHEN, ma soprattutto un tempo di QUERY() di gran lunga minore di PELLCAPP e CHEN, e che non presenta una crescita esponenziale. Questi enormi vantaggi, che lo portano ad essere un algoritmo eseguibile anche dove lo stato dell'arte non lo è, sono in parte bilanciati dal ratio (figura 4.15): esso in generale cresce all'aumentare di N, ma non supera un fattore pari a 1.3 per tutti i dataset tranne che per HIGGS, per cui PELLCAPPDELTA20 arriva ad ottenere un ratio di 2, ma PELLCAPPDELTA05 rimane molto valido.

In conclusione, utilizzare l'algoritmo PELLCAPPDELTAxx porta a prestazioni migliori sia in termini temporali che spaziali, senza peggiorare di troppo la qualità della soluzione calcolata, e permette di ottenere una soluzione anche su finestre in cui gli altri algoritmi non possono essere eseguiti.

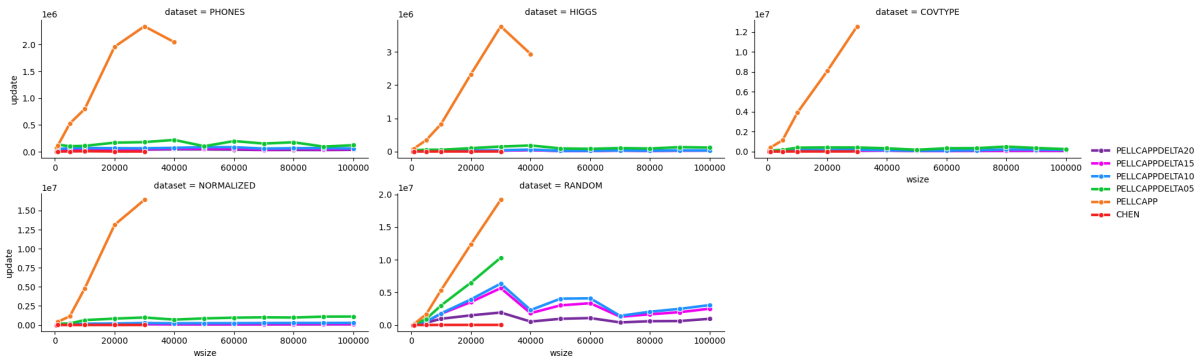

Figura 4.13: Tempo di esecuzione della procedura UPDATE(p) al variare della dimensione N della window

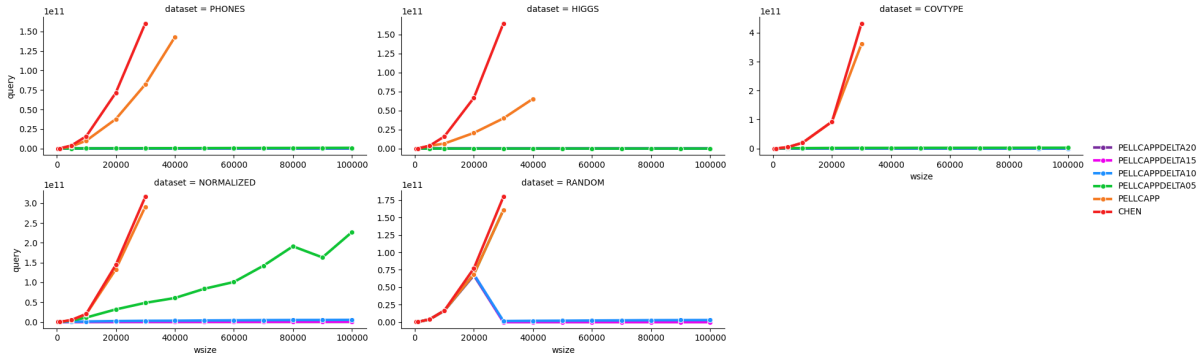

Figura 4.14: Tempo di esecuzione della procedura QUERY() al variare della dimensione N della window

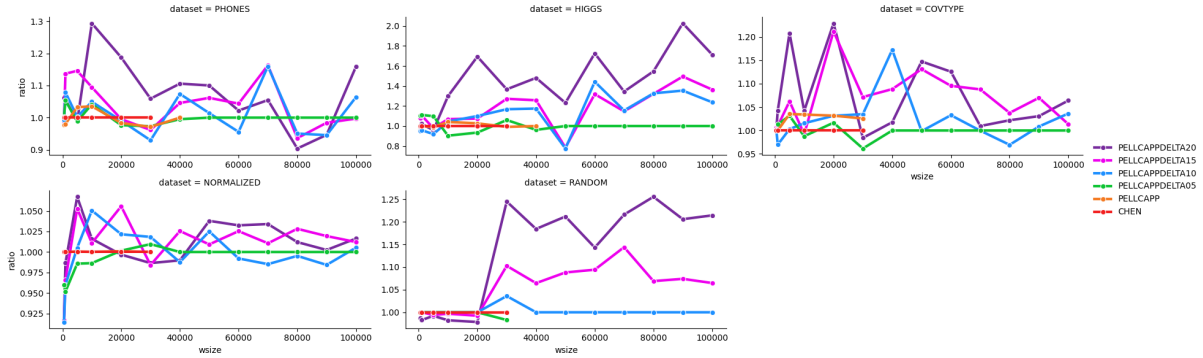

Figura 4.15: Ratio rispetto al miglior algoritmo al variare della dimensione N della window

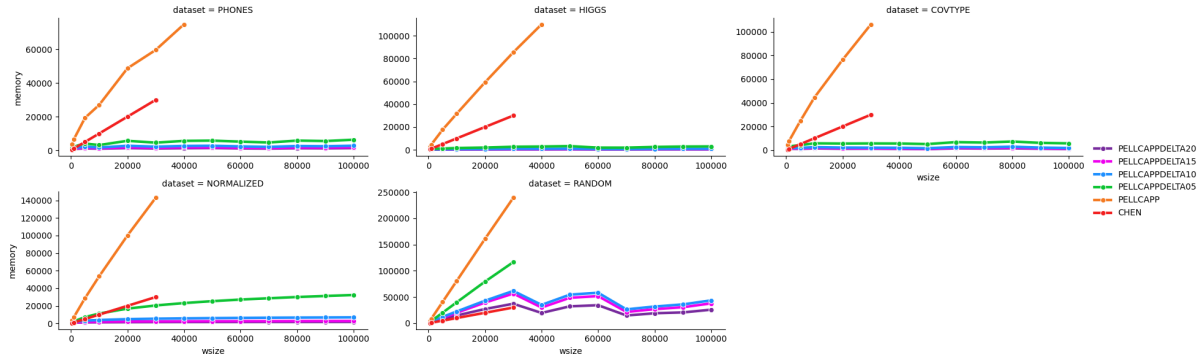

Figura 4.16: Numero di punti mantenuti in memoria al variare della dimensione N della window

4.2.3 Variazione di β

Il parametro β è un parametro di precisione come δ, che però influenza il numero e il tipo di guess degli algoritmi derivati da CAPP, in quanto è il valore principale della progressione geometrica di cui fanno parte tali guess. Aumentarlo significa aumentare la distanza tra i guess e diminuirne il numero, ottenendo in generale una soluzione meno precisa, guadagnando però in termini temporali e spaziali. Per quanto riguarda l'esecuzione di CHEN, essa non viene in alcun modo influenzata da tale parametro: la sua esecuzione al variare di β viene mostrata come indicazione delle condizioni della macchina in quel momento: si noti ad esempio in figura 4.18 come durante l'esecuzione di QUERY() nel dataset RANDOM per $\beta = 2$ il tempo di esecuzione sia stato leggermente minore rispetto al tempo per gli altri valori di β.

Come previsto, gli andamenti del numero di punti in memoria (figura 4.20) e del tempo di UPDATE(p) (figura 4.17) sono in diminuzione all'aumentare di β, fino a raggiungere un plateau (confermato anche per valori di β maggiori di 50, omessi per brevità), mentre in figura 4.18 si nota che il tempo di QUERY() non subisce significative variazioni per nessun algoritmo. Questo è giustificato dal fatto che cambiando β viene modificato il numero di guess fatti, ma una volta fissato il guess la costruzione del coreset T e l'esecuzione di CHEN su tale coreset rimangono invariati. In figura 4.19 è raffigurato il ratio rispetto a CHEN, che peggiora all'aumentare di β: utilizzando quindi un δ che faccia avere un ϵ molto grande e un β elevato viene degradata la qualità della soluzione. Si noti comunque che tale ratio non supera mai il valore 2, e talvolta (come nel caso di RANDOM) utilizzare un δ elevato può portare a un leggero miglioramento nella soluzione fornita.

In conclusione, l'utilizzo di un valore di β basso, pur comportando un numero di punti in memoria più alto, fornisce una soluzione più precisa. Per questo motivo il valore utilizzato di default è 2.

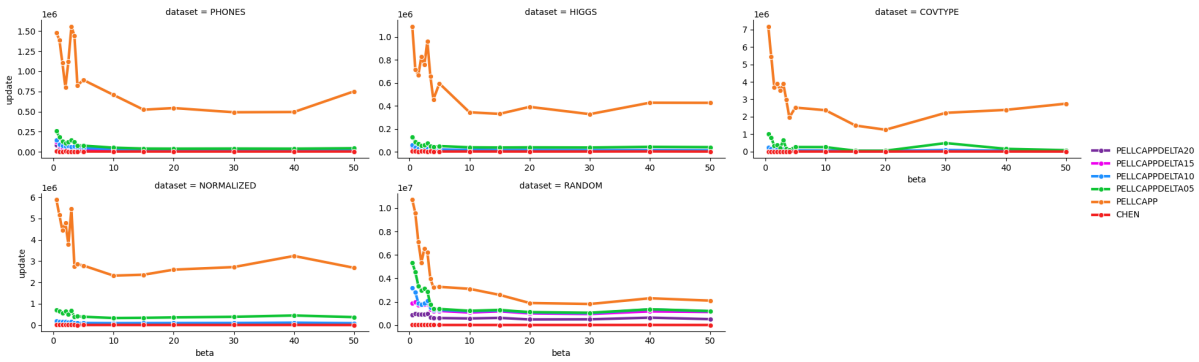

Figura 4.17: Tempo di esecuzione della procedura UPDATE(p) al variare di β

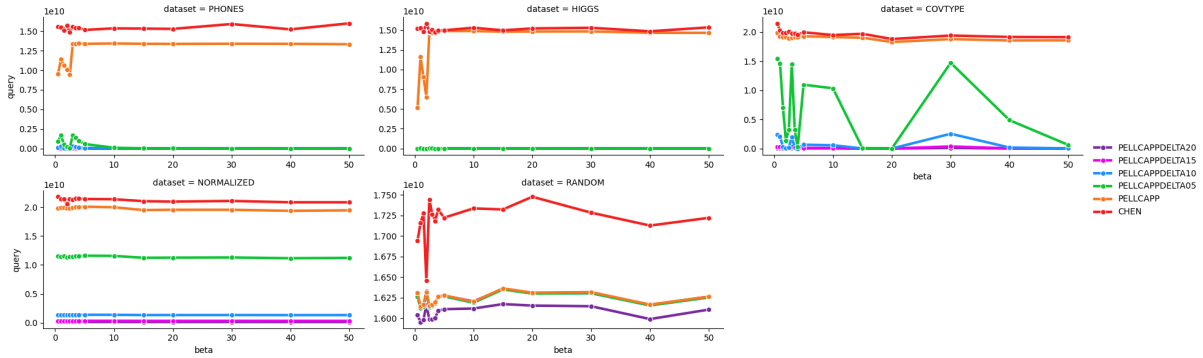

Figura 4.18: Tempo di esecuzione della procedura QUERY() al variare di β

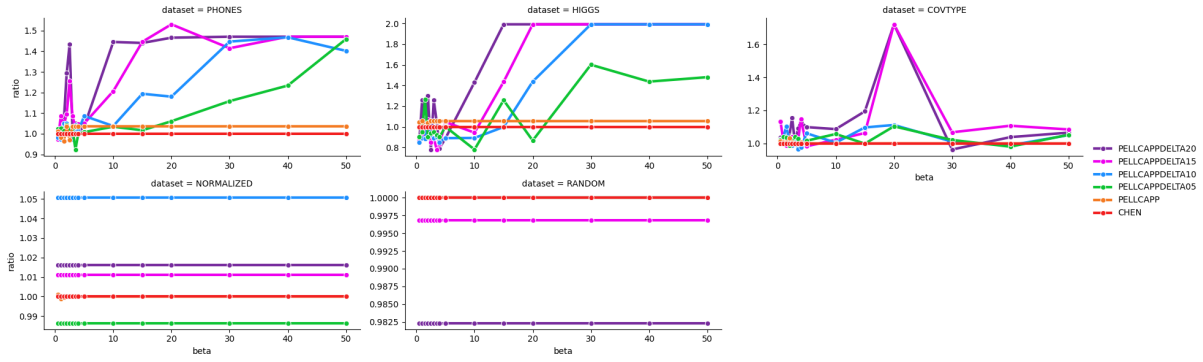

Figura 4.19: Ratio rispetto al miglior algoritmo al variare di β

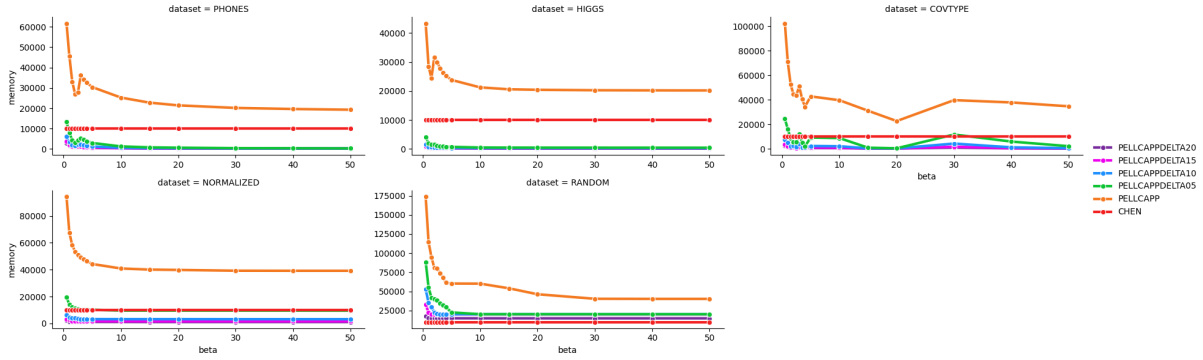

Figura 4.20: Numero di punti mantenuti in memoria al variare di β

4.2.4 Variazione di K

Il valore K indica il numero di cluster che si vogliono ottenere: per gli algoritmi derivati da CAPP, un suo aumento rende necessario mantenere più punti in memoria (come raffigurato in 4.25 la crescita è lineare), e questo porta anche PELLCAPPDELTAxx ad utilizzare una memoria maggiore di CHEN. Anche il tempo di UPDATE(p) (figura 4.21) aumenta linearmente

all'aumentare di K, superando il tempo di CHEN, ma questo viene compensato da un tempo di QUERY() (figura 4.22) che si mantiene spesso al di sotto di quello di CHEN, anche se al crescere di K tende a raggiungerlo più o meno velocemente a seconda del valore di δ e del dataset considerato. A seconda della frequenza con cui si fa tale query può quindi essere più o meno vantaggioso utilizzare PELLCAPPDELTAxx oppure CHEN. Il ratio rispetto a CHEN (figura 4.24), sebbene per valori di K piccoli tenda ad essere anche vicino a 1.3, con il crescere di K diminuisce, fino a stabilizzarsi attorno ad 1. Una spiegazione è che aumentare il numero di cluster permetta di includere con maggiore precisione quasi tutti i punti, e questo diminuisce di molto il raggio della soluzione trovata (figura 4.24), ma allo stesso tempo ciò significa che sono molti i guess validi in cui non viene limitato il numero di punti in OV_γ, O_γ e A_γ. Questo causa un aumento del tempo di UPDATE(p), e dato che il numero di punti del coreset cresce, cresce anche il tempo di QUERY().

Si può dunque concludere che all'aumentare del valore di K si assottiglino le differenze tra PELLCAPPDELTAxx e CHEN: se si vuole avere un minor utilizzo di memoria è conveniente usare quest'ultimo, mentre per ottenere l'insieme di centri in minor tempo è più conveniente il primo.

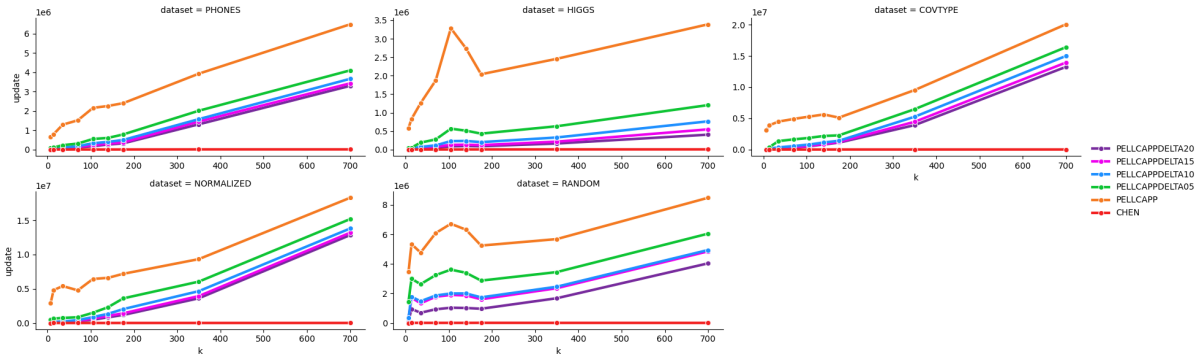

Figura 4.21: Tempo di esecuzione della procedura UPDATE(p) al variare di K

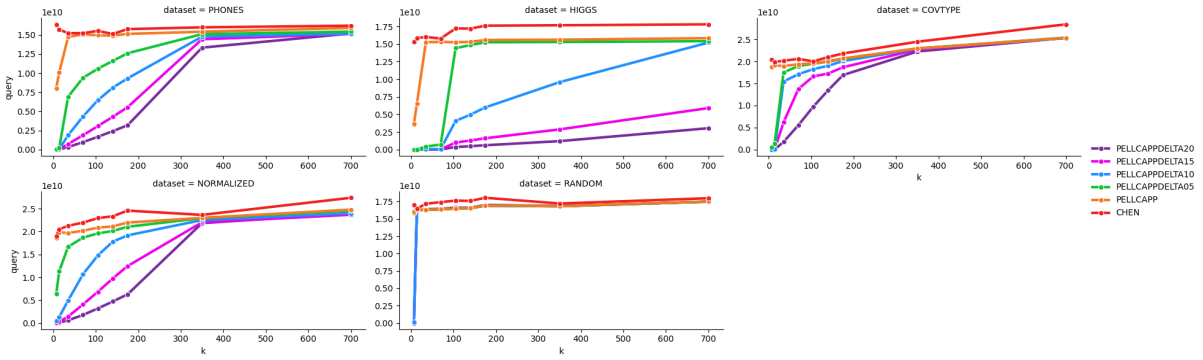

Figura 4.22: Tempo di esecuzione della procedura QUERY() al variare di K

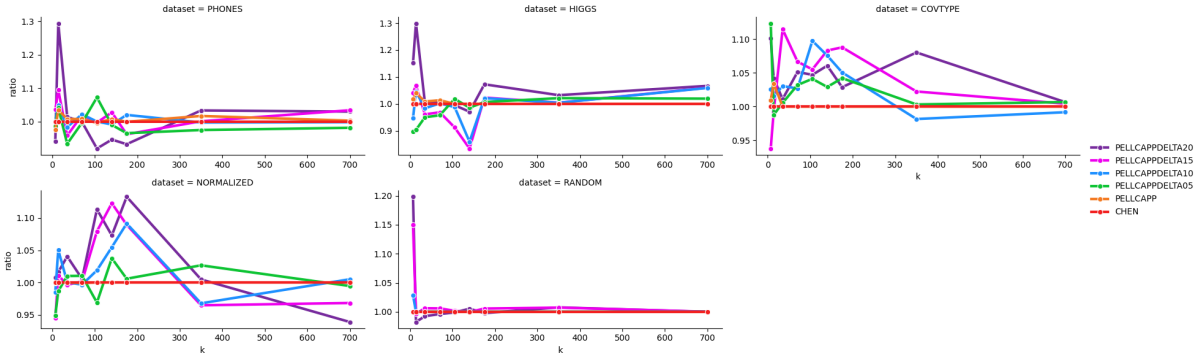

Figura 4.23: Ratio rispetto al migliore algoritmo al variare di K

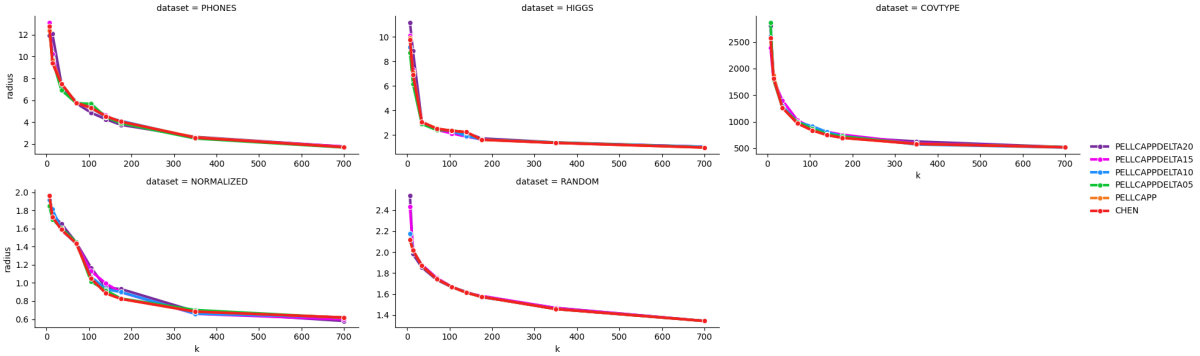

Figura 4.24: Raggio della soluzione trovata al variare di K

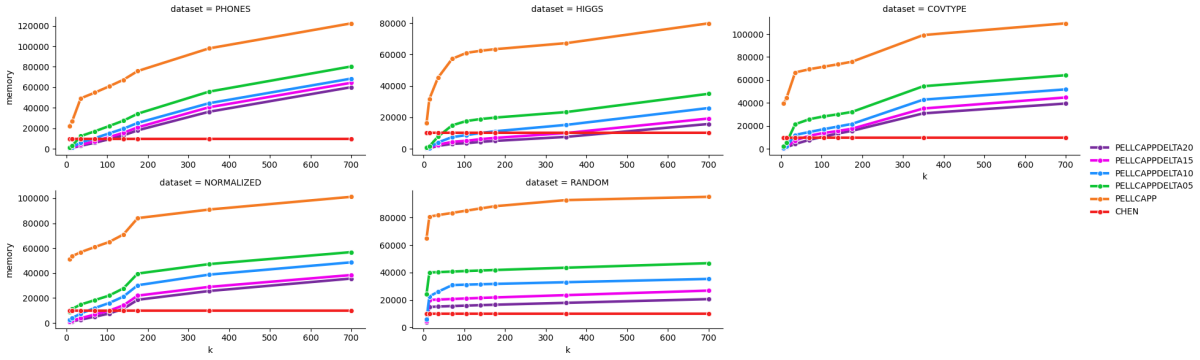

Figura 4.25: Numero di punti mantenuti in memoria al variare di K

4.2.5 Variazione della dimensionalità

L'ulitmo insieme di esperimenti è stato eseguito sui dataset BLOBS, già descritti nella Sezione Dataset. Variare la dimensionalità è più facile e più misurabile che variare la doubling dimension dello stream, la quale naturalmente viene modificata da tale variazione. E' importante notare

che, oltre alla dimensionalità, in questi dataset è presente anche una variazione dell'aspect ratio, cosa che potrebbe viziare i risultati.

Nella figura 4.30 si nota che all'aumentare della dimensionalità il numero di punti in memoria in generale aumenta, ma non significativamente, come il tempo di UPDATE(p) (figura 4.26) e QUERY() (figura 4.27), mentre il ratio (figura 4.28) non subisce rilevanti modifiche (sebbene il raggio in figura 4.29 aumenti, in quanto aumenta la distanza in generale tra i punti).

Si presti attenzione al picco presente nel dataset a dimensionalità 15: per spiegarlo si può notare che la dimensione N della window (10 000 punti) è piccola per risolvere un problema di K-Center Clustering con 20 dimensioni e 21 cluster, in quanto tali cluster saranno molto distanziati tra di loro. Per questo motivo, l'algoritmo è in grado di ottenere prestazioni migliori, ottenendo anche un ratio molto vicino ad 1.

Questo dimostra, assieme agli esperimenti finora condotti, che la dimensionalità non influenza pesantemente il comportamento di tali algoritmi: essi funzionano ugualmente bene in dataset ad alta e a bassa dimensionalità, con lievi differenze prestazionali.

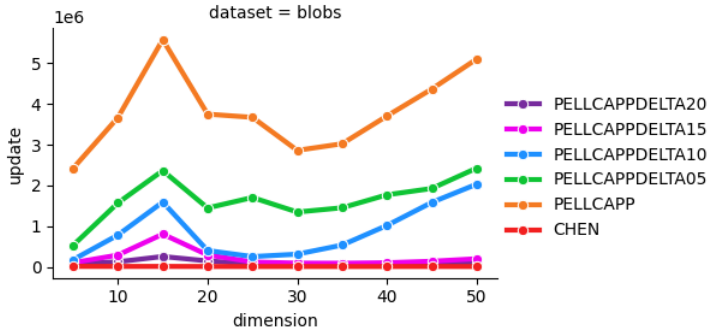

Figura 4.26: Tempo di esecuzione della procedura UPDATE(p) al variare della dimensionalità

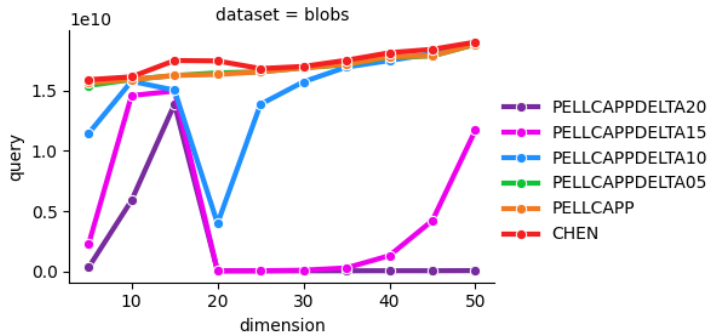

Figura 4.27: Tempo di esecuzione della procedura QUERY() al variare della dimensionalità

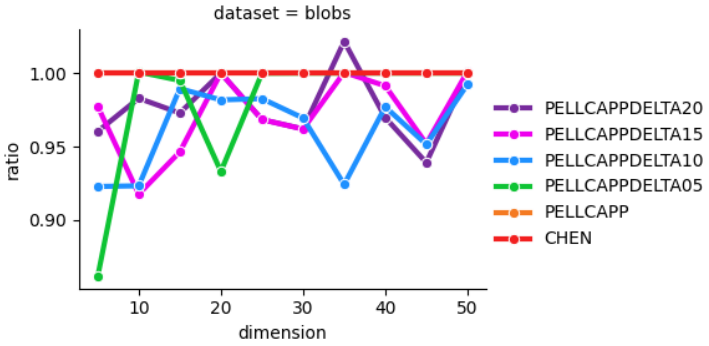

Figura 4.28: Ratio rispetto al miglior algoritmo al variare della dimensionalità

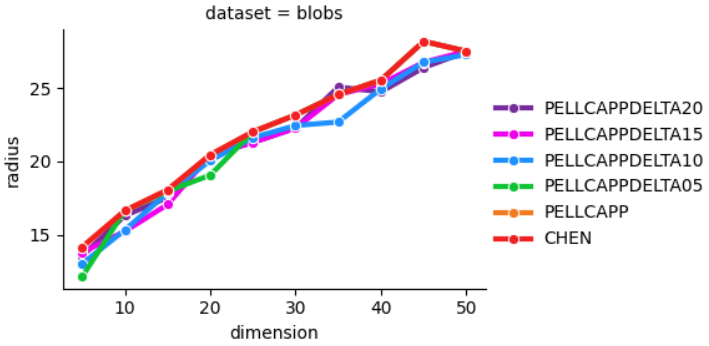

Figura 4.29: Raggio della soluzione trovata al variare della dimensionalità

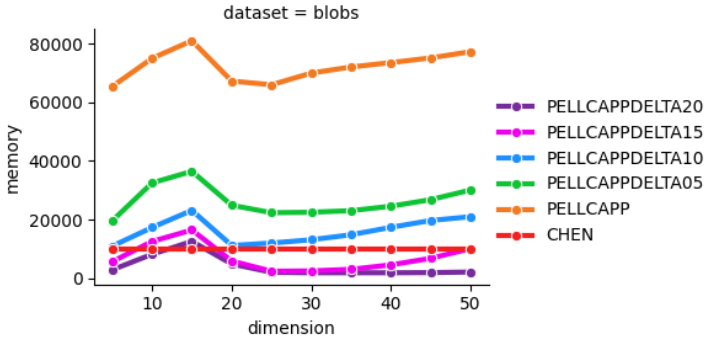

Figura 4.30: Numero di punti mantenuti in memoria al variare della dimensionalità

Capitolo 5

Conclusioni

L'analisi condotta in questo documento aveva un duplice scopo: da una parte si voleva confrontare la nuova strategia risolutiva del problema FKC nel modello Sliding Window di Cappellotto e al. [1] con l'algoritmo finora utilizzabile in tale modello, ovvero quello sequenziale di Chen e al. [2], e dall'altra osservare come si comportasse l'algoritmo di Cappellotto e al. [1] in funzione dei vari parametri che possono essere impostati per la sua esecuzione.

I risultati hanno evidenziato che gli algoritmi CAPPDELTAxx, in cui viene impostato il valore di δ invece che il corrispettivo valore di ϵ, sono molto più semplici da utilizzare rispetto all'originale CAPP, e presentano prestazioni temporali e spaziali migliori pur mantenendo una buona precisione. Questi algoritmi, e più precisamente le loro versioni oblivious COHCAPPDELTAxx e PELLCAPPDELTAxx, che non richiedono di conoscere a priori o calcolare i valori di massima e minima distanza, presentano in generale comportamenti di gran lunga migliori anche rispetto a CHEN: diminuendo il quantitativo di memoria richiesta riescono ad eseguire su finestre di grande ampiezza, mantenendo un tempo di UPDATE(p) comparabile con quello di CHEN e un tempo di QUERY() di molti ordini di grandezza inferiore. La qualità della soluzione, espressa come rapporto rispetto alla soluzione trovata da CHEN, non supera mai il valore 2 in nessuna configurazione considerata, e anzi spesso, anche con valori di δ e β elevati, si avvicina a 1. Se si vuole andare ad aumentare il valore di K, invece, tali algoritmi presentano una crescita lineare di memoria utilizzata, pur mantenendo il tempo di QUERY() al di sotto di quello di CHEN. La dimensionalità, invece, non influenza in maniera importante tali algoritmi: il loro comportamento con dataset ad alta dimensionalità è del tutto simile a quello con dataset a bassa dimensionalità, sebbene presentino dei leggeri peggioramenti di prestazioni temporali e spaziali.

Si può dunque concludere che l'idea di utilizzare gli insiemi coreset unitamente agli insiemi di validazione, derivata da Pellizzoni e al. [17] e Ceccarello e al. [20], consente di ottenere un grande vantaggio in numerose configurazioni rispetto a CHEN e la possibilità di ottenere

facilmente un compromesso tra la precisione della soluzione e l'utilizzo di tempo e spazio.

E' importante notare che questo lavoro non rappresenta un punto di fine per lo studio del problema FKC nel modello Sliding Window, ma una buona base da cui partire. In primo luogo, l'algoritmo di Cappellotto e al. [1] non affronta il problema FKC con outliers (Robust FKC), in cui un certo numero di punti può essere escluso dal calcolo del raggio della soluzione restituita. Nel modello Sliding Window non esistono ad oggi algoritmi efficienti per tale problema, seppure qualsiasi algoritmo applicabile nel modello sequenziale sarebbe utilizzabile anche in questo modello. Un buon punto di partenza per la risoluzione di questo problema potrebbe risiedere in Pellizzoni e al. [18]. Inoltre, il problema affrontato è relativo solamente all'utilizzo di un matroide di partizione, mentre non esistono ancora prove sperimentali per gli algoritmi che risolvono il problema del K-Center Clustering con il matroide trasversale descritti in Cappellotto e al. [1]. Più in generale, non esistono ad oggi algoritmi che risolvano nel modello Sliding Window il problema Matroid Center, in cui il matroide utilizzato è un generale matroide.

Ringraziamenti

Vorrei innanzitutto ringraziare i miei relatori, i professori Ceccarello, Pietracaprina e Pucci, che con gentilezza ed esperienza mi hanno aiutato a sentirmi meno inadeguato nel mondo della ricerca, e mi hanno offerto la possibilità di partecipare a questo interessante progetto.

Per tutti gli altri faccio una premessa: ho sempre trovato la scrittura dei ringraziamenti molto minimizzante. Come nella parte finale di un tema, si vuole condensare un insieme di esperienze, di momenti, di motivazioni in poche brevi frasi: non riuscirò a fare un tale riassunto, dato che sarebbe necessaria almeno una pagina per ognuno di voi.

Ringrazio innanzitutto i miei genitori per essere il mio punto di riferimento e la mia forza, e per avermi cresciuto garantendomi tutto ciò di cui un bambino ha bisogno, ovvero amore e supporto incondizionato. Grazie anche ai miei fratelli: essere cresciuto con voi, condividendo ricordi, risate, e qualche litigio, è il dono più grande. Grazie per essere sempre al mio fianco, in ogni passo del cammino. Grazie anche a Greta e Federico per essere diventati parte della nostra famiglia e per il loro sostegno, la loro gioia e la loro presenza.

Grazie a mio nonno e mia nonna per i valori che mi hanno insegnato quando ero piccolo: sono sicuro che siano molto fieri che il numero di laureati si sia alzato a dieci.

Ringrazio tutti i miei zii e cugini: grazie a Micio, Graziano, Silvia, Davide, Matteo, Valentina, Valeria, Alberto, Marta, Andrea, Anna e Luca, ma grazie anche ai miei zii della montagna, Andrea e Anna. Siete diventati il mio porto sicuro in cui trovare sempre calore e supporto.

Grazie a Marco, Michele, Lucia e Piero, compagni di mille avventure e vacanze, per i momenti di gioia che abbiamo vissuto come famiglia allargata.

Grazie a Emilio e Octavia per avermi accolto nella loro famiglia con affetto e generosità. Sono grato di poter condividere con voi la mia vita e di poter contare su di voi come se fossi un figlio. Ringrazio anche Anna, Antonio e Geta, che sebbene mi conoscano poco riescono sempre a farmi sentire a casa.

Ringrazio Federico, Andrea, Matteo, GianLuca, Davide I. e Davide P. per aver alleggerito questi ultimi anni, e aver condiviso studi, pranzi e sogni. Grazie per ogni chiacchierata e aiuto in tutto quello che ho fatto, so che su di voi potrò contare sempre.

Grazie a Nicolò per spingermi ogni giorno a migliorarmi e a guardare il mondo con occhio

scientifico, grazie a Matteo per tutti i momenti in cui mi ha dimostrato che si può puntare più in alto, e che in fondo non è così difficile raggiungere i propri sogni, e grazie ad Abebe per essere la mia spalla con cui condividere idee, consigli e risate.

Ringrazio infine i compagni con cui ho vissuto emozioni e ricordi che custodirò per sempre: grazie ad Enrico per la sua capacità di ascoltare e consigliare, a Denise per la sua inventiva e la sua estroversione, a Taibi per i suoi racconti e la sua capacità di rendere comica ogni banalità, a Gaia per il suo coraggio e la sua indipendenza, a Iby per la sua gentilezza e modestia, a Valentina per il suo modo leggero e divertente di vivere, ad Andrea per il suo sguardo artistico sul mondo.

E infine grazie a Larissa, che mi ha fatto diventare adulto, aggiustando i miei difetti senza farmi sentire sbagliato, consigliandomi anche quando non volevo ascoltare, e che è riuscita a darmi tutto quello di cui non sapevo nemmeno di aver bisogno. Con lei ho scoperto cosa significa amare e sentirsi amati: grazie per ogni istante passato insieme, per i tuoi gesti dolci e per la forza e complicità che troviamo l'uno nell'altro. Sei speciale, unica e insostituibile.

Grazie a tutti per quello che mi avete dato e che continuate a darmi, spero davvero che il nostro viaggio sia solo all'inizio.

Bibliografia

[1] L. Cappellotto, A. Pietacaprina e G. Pucci, «Matroid-center clustering in sliding windows,» tesi di laurea mag., Universita' degli studi di Padova, 2023. indirizzo: https://hdl.handle.net/20.500.12608/48142.

[2] D. Chen, J. Li, H. Liang e H. Wang, «Matroid and Knapsack Center Problems,» vol. 75, gen. 2013, isbn: 978-3-642-36693-2. doi: 10.1007/978-3-642-36694-9_10.

[3] M. R. Henzinger, P. Raghavan e S. Rajagopalan, «Computing on data streams,» in *External Memory Algorithms, Proceedings of a DIMACS Workshop, New Brunswick, New Jersey, USA, May 20-22, 1998*, J. M. Abello e J. S. Vitter, cur., ser. DIMACS Series in Discrete Mathematics and Theoretical Computer Science, vol. 50, DIMACS/AMS, 1998, pp. 107–118. doi: 10.1090/DIMACS/050/05. indirizzo: https://doi.org/10.1090/dimacs/050/05.

[4] M. Datar e R. Motwani, «The Sliding-Window Computation Model and Results,» in lug. 2016, pp. 149–165, isbn: 978-3-540-28607-3. doi: 10.1007/978-3-540-28608-0_7.

[5] W. Commons, *File:7 disks cover 2x disk.svg — Wikimedia Commons, the free media repository*, [Online; accessed 17-September-2024], 2023. indirizzo: https://commons.wikimedia.org/w/index.php?title=File:7_disks_cover_2x_disk.svg&oldid=771260306.

[6] E. Dandolo, A. Mazzetto, A. Pietracaprina e G. Pucci, «MapReduce Algorithms for Robust Center-Based Clustering in Doubling Metrics,» *Journal of Parallel and Distributed Computing*, vol. 194, p. 104 966, ago. 2024. doi: 10.1016/j.jpdc.2024.104966.

[7] A. Schrijver, *Combinatorial optimization polyhedra and efficiency*. Berlin: Springer, 2003. indirizzo: https://link.springer.com/book/9783540443896.

[8] D. Chen, J. Li e H. Wang, «Efficient Algorithms for One-Dimensional k-Center Problems,» *Theoretical Computer Science*, vol. 592, gen. 2013. doi: 10.1016/j.tcs.2015.05.028.

[9] T. F. Gonzalez, «Clustering to minimize the maximum intercluster distance,» *Theoretical Computer Science*, vol. 38, pp. 293–306, 1985, issn: 0304-3975. doi: https://doi.org/10.1016/0304-3975(85)90224-5. indirizzo: https://www.sciencedirect.com/science/article/pii/0304397585902245.

[10] D. Hochbaum e D. Shmoys, «A Best Possible Heuristic for the k-Center Problem,» *Mathematics of Operations Research - MOR*, vol. 10, pp. 180–184, giu. 1985. doi: 10.1287/moor.10.2.180.

[11] M. Charikar, S. Khuller, D. Mount e G. Narasimhan, «Algorithms for facility location problems with outliers,» gen. 2001, pp. 642–651. doi: 10.1145/365411.365555.

[12] R. McCutchen e S. Khuller, «Streaming Algorithms for k-Center Clustering with Outliers and with Anonymity,» in ago. 2008, pp. 165–178, isbn: 978-3-540-85362-6. doi: 10.1007/978-3-540-85363-3_14.

[13] S. Guha, «Tight results for clustering and summarizing data streams,» in *Proceedings of the 12th International Conference on Database Theory*, ser. ICDT '09, St. Petersburg, Russia: Association for Computing Machinery, 2009, pp. 268–275, isbn: 9781605584232. doi: 10.1145/1514894.1514926. indirizzo: https://doi.org/10.1145/1514894.1514926.

[14] M. Ceccarello, A. Pietracaprina e G. Pucci, *Solving k-center Clustering (with Outliers) in MapReduce and Streaming, almost as Accurately as Sequentially*, 2021. arXiv: 1802.09205 [cs.DC]. indirizzo: https://arxiv.org/abs/1802.09205.

[15] D. E. Knuth, *The TEX Book*. Addison-Wesley Professional, 1986.

[16] V. Cohen-Addad, C. Schwiegelshohn e C. Sohler, «Diameter and k-Center in Sliding Windows,» in *43rd International Colloquium on Automata, Languages, and Programming (ICALP 2016)*, I. Chatzigiannakis, M. Mitzenmacher, Y. Rabani e D. Sangiorgi, cur., ser. Leibniz International Proceedings in Informatics (LIPIcs), vol. 55, Dagstuhl, Germany: Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2016, 19:1–19:12, isbn: 978-3-95977-013-2. doi: 10.4230/LIPIcs.ICALP.2016.19. indirizzo: https://drops.dagstuhl.de/entities/document/10.4230/LIPIcs.ICALP.2016.19.

[17] P. Pellizzoni, A. Pietracaprina e G. Pucci, «Adaptive k-center and diameter estimation in sliding windows,» *International Journal of Data Science and Analytics*, vol. 14, n. 2, pp. 155–173, 2022, issn: 2364-4168. doi: 10.1007/s41060-022-00318-z. indirizzo: https://doi.org/10.1007/s41060-022-00318-z.

[18] P. Pellizzoni, A. Pietracaprina e G. Pucci, «k-Center Clustering with Outliers in Sliding Windows,» *Algorithms*, vol. 15, n. 2, 2022, issn: 1999-4893. doi: 10.3390/a15020052. indirizzo: https://www.mdpi.com/1999-4893/15/2/52.

[19] M. de Berg, M. Monemizadeh e Y. Zhong, «k-Center Clustering with Outliers in the Sliding-Window Model,» in *29th Annual European Symposium on Algorithms (ESA 2021)*, P. Mutzel, R. Pagh e G. Herman, cur., ser. Leibniz International Proceedings in Informatics (LIPIcs), vol. 204, Dagstuhl, Germany: Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021, 13:1–13:13, isbn: 978-3-95977-204-4. doi: 10.4230/LIPIcs.ESA.2021.13. indirizzo: https://drops.dagstuhl.de/entities/document/10.4230/LIPIcs.ESA.2021.13.

[20] M. Ceccarello, A. Pietracaprina, G. Pucci e F. Soldà, «Scalable and space-efficient Robust Matroid Center algorithms,» *Journal of Big Data*, vol. 10, apr. 2023. doi: 10.1186/s40537-023-00717-4.

[21] D. Chakrabarty e M. Negahbani, «Generalized Center Problems with Outliers,» *ACM Trans. Algorithms*, vol. 15, n. 3, 2019, issn: 1549-6325. doi: 10.1145/3338513. indirizzo: https://doi.org/10.1145/3338513.

[22] A. Chiplunkar, S. Kale e S. N. Ramamoorthy, «How to Solve Fair k-Center in Massive Data Models,» in *Proceedings of the 37th International Conference on Machine Learning*, H. D. III e A. Singh, cur., ser. Proceedings of Machine Learning Research, vol. 119, PMLR, 2020, pp. 1877–1886. indirizzo: https://proceedings.mlr.press/v119/chiplunkar20a.html.

[23] D. Michail, J. Kinable, B. Naveh e J. Sichi, *JGraphT – A Java library for graph data structures and algorithms*, apr. 2019.

[24] H. Blunck, S. Bhattacharya, T. Prentow, M. Kjrgaard e A. Dey, *Heterogeneity Activity Recognition*, UCI Machine Learning Repository, DOI: https://doi.org/10.24432/C5689X, 2015.

[25] D. Whiteson, *HIGGS*, UCI Machine Learning Repository, DOI: https://doi.org/10.24432/C5V312, 2014.

[26] J. Blackard, *Covertype*, UCI Machine Learning Repository, DOI: https://doi.org/10.24432/C50K5N, 1998.