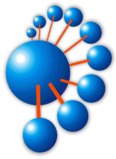
UNIVERSIDAD DE CONCEPCIÓN
FACULTAD DE INGENIERÍA
DEPARTAMENTO DE INGENIERÍA INFORMÁTICA
Y CIENCIAS DE LA COMPUTACIÓN



Estimación de medidas de monitoreo de red mediante sketches basados en cuantiles

Alumno
Lucas Gonzalo Kraemer Ananías

Profesor Guía
Cecilia Hernández
Profesor Co-Guía
Miguel Figueroa

Concepción, agosto 2024

Resumen

En la actualidad el monitoreo de una red permite determinar el desempeño, estado de congestión y la detección de actividades maliciosas. Para ello, se necesita computar estadísticas del tráfico de red. Por ejemplo, se puede diferenciar tráfico normal de anómalo caracterizando el estado de la red de datos, distribución de flujos y tamaño de datos de los flujos. Los principales desafíos en el cómputo de estas estadísticas en grandes cantidades de datos es el requerimiento espacial y tiempo de procesamiento. Para afrontar esta problemática se han utilizado sketches, los cuales corresponden a estructuras reducidas en espacio que almacenan información que permite resolver consultas específicas respecto a las estadísticas de los datos, comúnmente usados en línea y que requieren espacio sublineal para entregar resultados aproximados.

La presente memoria de título implementa dos sketches basados en cuantiles, MRL y KLL con el fin de estimar características en los flujos de una red. Adicionalmente, se propone una adaptación a los sketches desarrollados que procesa una mayor cantidad de información, la cual es utilizada para estimar los top-K flujos con mayor payload pertenecientes a una red. Se compara el rendimiento de las estructuras desarrolladas con un sketch existente en la literatura, donde se encuentra que existe espacio para mejora en los sketches implementados. Aún así, los valores de precisión en la estimación de los top-K son mejores en los sketches implementados, llegando a precisiones de hasta 0,67.

Índice general

Índice de tablas

Índice de figuras

Capítulo 1

Introducción

Los avances en el desarrollo de tecnologías en redes de datos ha permitido el aumento progresivo en las velocidades de comunicación. Es así como actualmente es posible encontrar enlaces de hasta 400 Gbps [1]. La gestión de administración de la red típicamente requiere del cómputo de distintas estadísticas como la latencia, volumen del tráfico y pérdida de paquetes, las cuales se emplean en el monitoreo del desempeño, el estado de congestión y detección de actividades maliciosas. Tales estadísticas pueden ser determinadas haciendo un seguimiento de las trazas de red. Estas trazas almacenan flujos de datos en forma de secuencias de paquetes que tienen atributos comunes como IP de origen y destino; protocolos de la capa de transporte como TCP o UDP; y puertos de origen y destino [2, 3]. Mediante el análisis de estas estadísticas se puede resolver consultas acerca del comportamiento del flujo de red en tiempo real. Sin embargo, el cómputo exacto de estadísticas como flujos con mayor payload y distribución de flujos requieren un espacio lineal y tiempos de respuestas más altos, lo que dificulta proporcionar respuestas en tiempo real en dispositivos de red con baja capacidad de almacenamiento tales como switches.

Para solucionar estos problemas, se han usado distintas técnicas entre las cuales se encuentra el *muestreo* [4], que busca reducir el tiempo y costo de análisis mediante la selección de una parte de un conjunto para poder obtener información respecto a su totalidad. Aún así, se debe decidir entre tener una alta precisión o tener altas tasas de muestreo, las cuales pueden llegar a consumir una gran cantidad de recursos. Otra alternativa consiste en el uso de *sketches* [4], los cuales corresponden a estructuras reducidas en espacio que requieren espacio sublineal, las cuales almacenan información que permite resolver consultas específicas respecto a las estadísticas de los datos, comúnmente usados en línea y que entregan resultados aproximados. Existen distintos tipos de sketches según el problema y consulta a realizar, como los asociados al problema de estimación de elementos distintos o cardinalidad de un conjunto; los sketches de cuentas que estiman el número de ocurrencia de los elementos dentro de un conjunto; de la misma manera, existen sketches para determinar los cuantiles de un conjunto de datos, es decir, encontrar los elementos que dividen una distribución en cierta proporción, como por ejemplo la mediana. Algunas implementaciones de sketches para los problemas mencionados corresponden a *HyperLogLog*(HLL) [5] para cardinalidad, *Count Min*(CM) [6] y *Count Min with Conservative Upgrades*(CM-CU) [7] para cuentas y *KLL* [8] para cuantiles.

Existen distintos tipos de modelos que indican las operaciones que soporta cada sketch. El modelo *insertion-only* solo permite insertar elementos, mientras que el modelo *turnstile* indica que todo elemento insertado puede ser posteriormente eliminado [9] y el *bounded deletion model* indica que hasta cierta proporción dada de los elementos insertados pueden ser eliminados [10]. Un ejemplo de un sketch que trabaja con el *bounded deletion model* es el sketch $KLL^{\pm}$ [10].

La presente memoria de título consiste en la implementación de los sketches MRL y KLL para la estimación de cuantiles de red. Además, se diseña e implementa una estrategia para estimar el payload por flujo basado en estos sketches. La estrategia implementada está alineada con la idea de sketches reversibles, cuyo objetivo es identificar los elementos del stream y no solo las características de ellos. Para la evaluación experimental se utilizan trazas reales y se emplean métricas basadas en error relativo, precisión y recall, como también en el uso de recursos tales como el tiempo de cómputo de las operaciones y el espacio utilizado.

1.1. Objetivo General

Este trabajo tiene como objetivo el diseño e implementación de estrategias que utilicen sketches basados en cuantiles para estimar frecuencias, payloads y los top-k flujos en el monitoreo del tráfico de la red.

1.1.1. Objetivos Específicos

1. Diseñar e implementar un algoritmo basado en sketches para estimar la distribución de las frecuencias de los flujos en una red.
2. Diseñar e implementar un algoritmo basado en sketches para estimar los top-K flujos con mayor payload pertenecientes a una red.
3. Evaluar el desempeño y precisión de las alternativas desarrolladas.

Capítulo 2

Revisión bibliográfica

2.1. Sketches

El término sketch hace referencia a una estructura de datos reducida en espacio, la cual permite almacenar un resumen de la información relevante de los datos para evaluar y/o calcular una estimación de una función específica. Es importante destacar que en muchos casos los sketches no almacenan los elementos de forma explícita, sino que emplean distintas técnicas para mantener su información obteniendo así espacios sublineales. Estas estructuras son principalmente utilizadas en algoritmos de streaming, donde se requiere el procesamiento de grandes cantidades de datos para responder consultas, idealmente durante el procesamiento del stream.
Algunas operaciones que pueden realizar consisten en la actualización y registro de la información ingresada, la consulta para proporcionar una estimación de la función específica del sketch, la unión para fusionar varios sketches en uno solo y la recuperación para obtener de forma inversa un elemento que fue previamente registrado [2, 11]. El método empleado para realizar cada operación distingue entre un sketch y otro, por lo que pueden existir varios sketches que realicen una misma tarea diferenciándose en la forma que almacenan la información, la calidad de sus estimaciones y el espacio empleado para dicha tarea. Asimismo, existen tres modelos que indican cuáles son las capacidades en la etapa de la actualización de la información. El primero consiste en el insertion only model, también conocido como caja registradora, en el que los datos solo pueden ser ingresados por el stream. El turnstile model indica que se pueden soportar operaciones tanto de ingreso como de eliminación, con la condición de que todo elemento eliminado debió ser previamente ingresado. El último modelo consiste en el bounded deletion model, el cual se asemeja bastante al turnstile model pero con la limitación de que solo se puede eliminar hasta una cierta proporción dada de los elementos [10]. La mayoría de los sketches trabaja en el insertion only model, principalmente debido a la complejidad asociada a quitar la información que ya se encuentra resumida en la estructura.

Algunas funciones tradicionales o más comunes que estiman los sketches consisten en:

- La cardinalidad de un multi-conjunto: Se busca obtener la cantidad de elementos distin-

tos encontrados en un conjunto de datos. Uno de los sketches más conocidos que trabaja este problema es HyperLogLog [5], el cual emplea el uso de un arreglo y una función hash para obtener una cota en espacio $\mathcal{O}(log(log(n)))$. Otros sketches que determinan la cardinalidad son LogLog [12], y SuperSketch [13].

- La estimación de cuentas: Se quiere determinar el número de ocurrencias de un determinado elemento. El sketch Count Min (CM) [6] mantiene cuentas mediante el uso de contadores y funciones hash. Count Min with Conservative Upgrades (CM-CU) [14] y CountSketch [15] consisten en alternativas para resolver este problema.

Generalmente el uso de algoritmos de streaming empleando sketches contiene errores asociados, ya que estos son aproximados y/o aleatorizados. Es por ello que se establece una manera para que el usuario determine los parámetros de acuerdo a una cota de error tolerada ε, y la confiabilidad que ésta otorga δ. En otras palabras, se desea que con cierta probabilidad los valores obtenidos no superen el error de estimación de acuerdo a la cota definida por el sketch.

2.2. El problema de Cuantiles

Un cuantil hace referencia a un punto encontrado en una función de distribución de una variable aleatoria, que separa los elementos en una proporción. Específicamente, se denomina a un cuantil de orden p a aquel valor de la población que es mayor o igual al p porcentaje de los datos. Por ejemplo, un cuantil de orden 0,3 tendrá un 30 % de la totalidad de los datos con un valor menor o igual al de dicho cuantil, asimismo, el cuantil de orden 0,5 corresponderá a la mediana. En la Figura 2.2.1 se encuentra un ejemplo para visualizar distintos cuantiles.

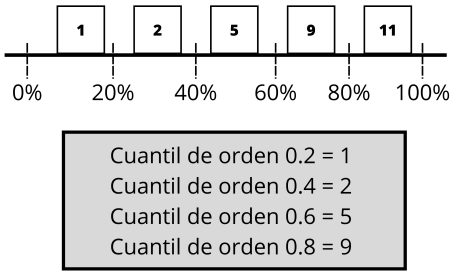


Figura 2.2.1: Visualización de distintos cuantiles de orden p.

Comúnmente los cuantiles son empleados para obtener grupos que dividen la distribución en partes iguales, donde cada intervalo contiene la misma proporción de valores que el resto [16]. Algunos de los grupos más populares corresponden a cuartiles, quintiles, deciles y percentiles, los cuales dividen los datos en grupos de 4, 5, 10 y 100 partes iguales respectivamente. El uso de cuantiles permite comprender mejor la dispersión, la forma de la distribución, segmentar datos y la comparación entre dos o más distribuciones. En la Figura 2.2.1 se puede apreciar

la extracción de los quintiles, pues se dividen los datos según los cuantiles de orden 0.2, 0.4, 0.6 y 0.8.

Es conocido que no es posible calcular cuantiles exactos por medio de una única pasada de los datos sin almacenarlos todos, pues requiere un espacio de al menos $\mathcal{O}(n)$ [17]. Aún así, en varias aplicaciones no es necesario encontrar el valor exacto de un cuantil, sino que se puede tolerar una aproximación del valor verdadero. En estos casos se establece una cota de error de estimación aceptada denominada ε.

Matemáticamente, dado un set de elementos $x_1, ..., x_n$ en un flujo o stream con orden arbitrario, el valor asociado al cuantil de un elemento x corresponde a la fracción de elementos en el stream que cumplen $x_i \leq x$ [8]. En este contexto, se define el rank de x como $R(x)$, el cual indica el número de elementos que cumplen la condición mencionada. Por motivos tales como memoria limitada o rapidez, es posible que se desee obtener una aproximación de un cuantil, el cual tolera un error determinado por un parámetro ε proporcionado. Específicamente, dado un set de elementos $x_1, ..., x_n$, se desea obtener una estimación del rank de un elemento x, $\hat{R(x)}$, en el que se cumpla que $|\hat{R(x)} - R(x)| \leq \varepsilon n$ con una probabilidad de $1 - \delta$. Respecto a ello, se propone resolver un problema denominado *all quantile problem*, el cual construye una estructura de datos que computa $\hat{R(x)}$, donde con probabilidad $1 - \delta$ simultáneamente para todos los valores se cumple que $|\hat{R(x)} - R(x)| \leq \varepsilon n$.

Algunos sketches que pueden proporcionar estimaciones del *all quantile problem* corresponden al MRL [18] y al KLL [8], estos pueden responder consultas respecto a las siguientes operaciones:

1. Rank(elemento x):
   Dado un elemento x, se desea obtener el número de elementos en el stream que cumplan que $x_i \leq x$.

2. Select(rank r):
   Dado un rank r, se desea obtener el mayor elemento en el stream tal que $Rank(x) \leq r$. Corresponde a la operación inversa de rank.

3. Quantile(cuantil q):
   Se desea obtener el elemento asociado al cuantil de orden q, el cual divide aproximadamente los datos en una proporción de q y $1 - q$. Esta consulta es equivalente a realizar la operación select(qn_i), donde n_i es la cantidad de datos procesados hasta el momento.

Estos sketches emplean una técnica basada en el uso de compactores, la cual será explicada a continuación.

2.2.1. Compactor

Un compactor puede ser pensado como un arreglo que puede almacenar hasta k elementos, donde cada uno tiene un peso asociado w. Cuando este arreglo se llena se realiza una operación denominada compactación, la cual procesa los elementos con el fin de entregar $\frac{k}{2}$ de estos con un peso asociado de $2w$. Específicamente, la compactación consiste en los siguientes pasos:

1. Se ordenan los k elementos encontrados en el compactor.
2. Se obtienen $\frac{k}{2}$ elementos mediante una selección aleatoria entre los elementos encontrados en posiciones pares o impares del compactor ordenado.
3. Los elementos que no fueron seleccionados son descartados.
4. El compactor se vacía y los elementos seleccionados son entregados como salida de la compactación con un peso respectivo al doble del compactor usado, correspondiendo a $2w$.

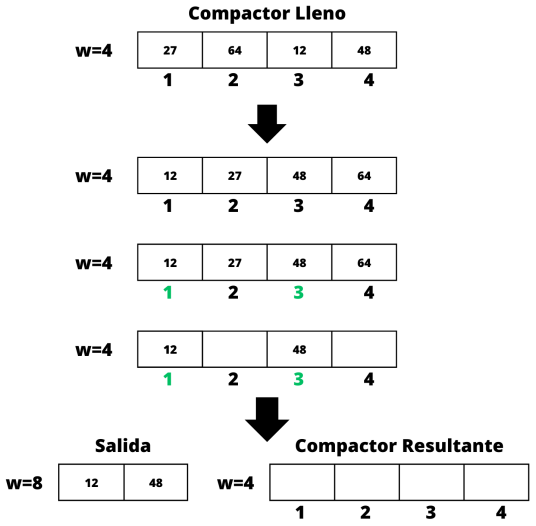


Figura 2.2.2: Ejemplo del procedimiento de una compactación.

Un ejemplo de este procedimiento se encuentra ilustrado en la Figura 2.2.2. Para mantener un registro de los elementos compactados y disminuir el espacio total ocupado, se ingresan los resultados de una compactación en un nuevo compactor con pesos asociados de $2w$. Este proceso de incorporar nuevos compactores se puede seguir realizando para procesar todos los elementos posibles. La Figura 2.2.3 indica el procedimiento a realizar cuando se realizan dos compactaciones, donde se incorpora un nuevo compactor para almacenar los resultados obtenidos.

Los compactores permiten aproximar consultas de tipo rank, donde se pueden observar diferencias en los valores anteriores y posteriores a una compactación. En el primer caso se poseen todos los elementos para poder responder correctamente la consulta, pero en el segundo caso solo se mantiene la mitad de los elementos, por lo que se puede introducir un error en la respuesta proporcionada. Este error corresponde a lo más a w, puesto que al eliminar

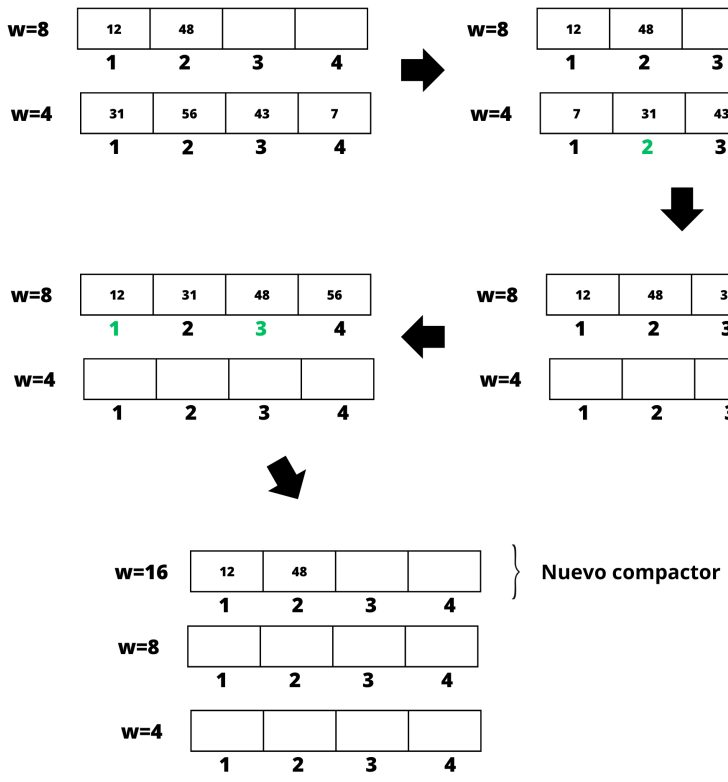


Figura 2.2.3: Ejemplo del procedimiento de una compactación que conlleva a otra.

un dato se garantiza la mantención de su antecesor y sucesor. Un ejemplo de lo mencionado se encuentra en la Figura 2.2.4, donde se pregunta por el rank de dos elementos distintos. En la primera fila se encuentran representados todos los elementos con un peso w previo a la compactación y las siguientes dos filas representan a las distintas posibilidades de los elementos, cada uno con peso $2w$. Además, se puede apreciar una diferencia entre los ranks estimados dependiendo del valor consultado y cuales elementos fueron eliminados en la etapa de compactación.

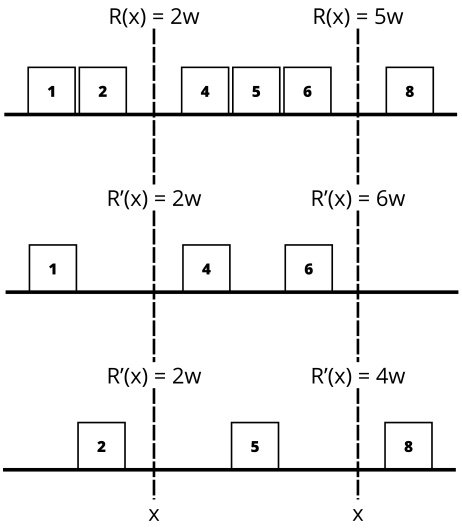


Figura 2.2.4: Ejemplo de un único compactor con 6 elementos realizando una compactación. La consulta de rank asociada a un elemento no cambia si el rank de dicho elemento en el compactor corresponde a una posición par, en caso contrario el rank indicado puede aumentar o disminuir en w. Fuente: Optimal Quantile Approximation in Streams [8].

2.2.2. Sketch MRL

El Manku, Rajagopalan and Lindsay sketch (MRL) [18] emplea el uso de varios compactores para analizar y procesar información para estimar cuantiles. En esta estructura todos los compactores pueden almacenar hasta k elementos, diferenciándose mediante una jerarquía de niveles, en los que cada compactor tiene asociado pesos distintos. Cada nivel compactará su contenido hacia el compactor de siguiente nivel, con tal que la información resultante sea guardada. La cantidad total de niveles H depende del parámetro k usado, donde para procesar una cantidad de n elementos provenientes del stream se establece que $H = \log(\frac{n}{k})$. Un ejemplo de visualización de la estructura del MRL se encuentra en la Figura 2.2.5.

El error asociado a una estimación proveniente del MRL depende del número de compactaciones realizadas. Concretamente, al realizar una compactación ubicada en un nivel h, se ingresa un error de a lo más $w = \pm 2^h$. La literatura [19] indica que la cantidad máxima de niveles a emplear es de a lo mas $2k\varepsilon$ para cumplir con la cota de error ε, donde se selecciona $k = \mathcal{O}(\varepsilon^{-1} \log(\varepsilon n))$. Se puede determinar la cantidad de espacio que ocupa el MRL considerando el número de compactores H y el tamaño de cada uno k, donde el espacio es $\mathcal{O}(kH) = \mathcal{O}(\varepsilon^{-1} \log^2(\varepsilon n))$.

Es importante destacar que el MRL soporta la unión entre dos o mas sketches, un algoritmo de sketch soporta esta operación cuando: Dado dos sketches S_1 y S_2 creados a partir de las entradas D_1 y D_2, permiten generar un sketch S de $D = D_1 \cup D_2$ manteniendo el error y probabilidad de fallo, bajo las mismas restricciones empleadas en S_1 y S_2 [7]. En muchos casos se desea que los sketches puedan soportar la unión, ya que generalmente grandes cantidades

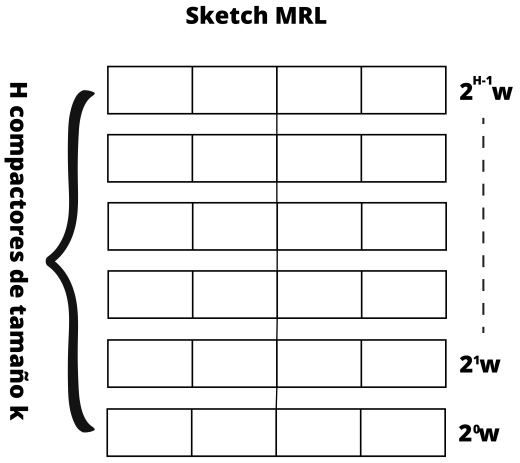


Figura 2.2.5: Visualización general de un sketch MRL.

de datos son trabajados por múltiples máquinas simultáneamente [8].

2.2.3. Sketch KLL

El Karnin, Lang and Liberty sketch (KLL) [8], al igual que el MRL, utiliza compactores para realizar su estimación. Este sketch busca optimizar el uso de espacio, proporcionando una mayor relevancia a aquellos compactores que se encuentran en un nivel superior, pues allí se encuentran los elementos con mayor peso.

Este sketch puede ser entendido como distintos cambios o mejoras respecto al MRL. El primero de estos cambios consiste en modificar la capacidad de los compactores en diferentes niveles, donde cada nivel soporta hasta una cantidad de k_h elementos y el compactor de mayor nivel poseerá hasta k_H elementos. Se ha encontrado que k_h puede disminuir exponencialmente sin afectar el valor de la cota de error asintótica [8]. Por ejemplo establecer $k_h \approx k_H(\frac{2}{3})^{H-h}$ ha tenido buenos resultados, mejorando así la complejidad espacial de la estructura. Para que un compactor funcione correctamente debe contener una capacidad mínima de 2, ya que es necesario poder realizar la compactación. La complejidad espacial del KLL con esta modificación es $\mathcal{O}(H) = \mathcal{O}(\log(\frac{n}{k}))$, la cual se puede mejorar. Al reducir exponencialmente la capacidad de los compactores, llegará un momento en que se encuentren H'' de estos con $k_h = 2$ en los que, al comparar la funcionalidad de todos los compactores de capacidad 2, resulta equivalente a hacer un muestreo de uno por cada $2^{H''}$ elementos. Reemplazando los H'' primeros compactores por una técnica de muestreo, ya no se necesitará de $\mathcal{O}(H'')$ espacio, sino de $\mathcal{O}(1)$. Luego, la capacidad de todos los compactores estará acotada por $\sum_{h=H''+1}^{H} k(\frac{2}{3})^{H-h} \leq 3k$, obteniendo una cota de $\mathcal{O}(k)$. Para cumplir con la cota de error se establece $k = \mathcal{O}((\frac{1}{\varepsilon})\sqrt{\log(1-\delta)})$. Una representación de este cambio se encuentra en la Figura 2.2.6.

El segundo cambio proviene del tratado especial que se le da a los $\log(\log(\frac{1}{\delta}))$ compactores de

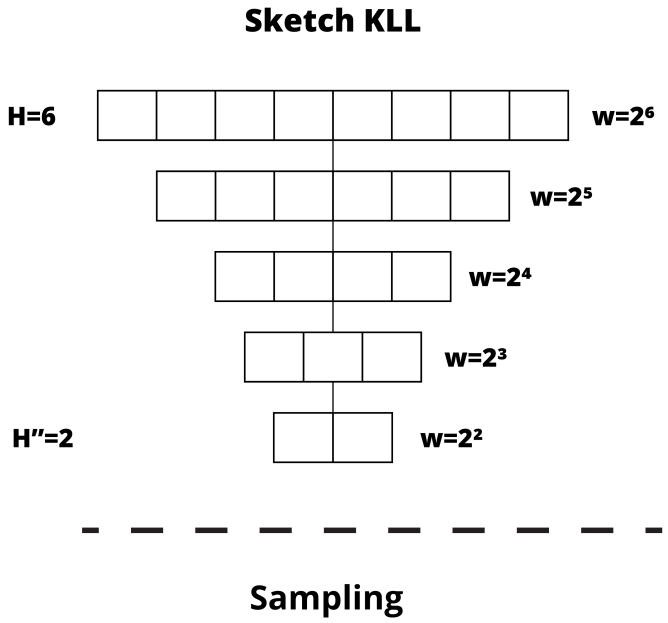


Figura 2.2.6: Ejemplo de estructura KLL, en donde solo se encuentra modificada la cantidad de elementos soportados por los compactores en cada nivel. La capacidad de cada compactor va disminuyendo exponencialmente.

mayor nivel. Esta modificación consiste en que a los primeros $\log(\log(\frac{1}{\delta}))$ compactores se les establece una capacidad correspondiente a k_H, mientras que el resto continua su disminución exponencial como se mencionaba anteriormente. Así, queda lo siguiente:

$$k_h = \begin{cases} k_H & \text{si } h \geq H - \log(\log(\frac{1}{\delta})); \\ k_H(\frac{2}{3})^{H-h} & \text{en caso contrario.} \end{cases}$$

Incorporando estos cambios, se puede separar el análisis de la parte superior del de la inferior, en los que se llega a una cota de $\mathcal{O}((\frac{1}{\varepsilon})\log^2\log(\frac{1}{\delta}))$ [8]. Con estas modificaciones, el sketch también soporta la operación de la unión y su estructura se verá como aquella encontrada en la Figura 2.2.7.

La última modificación consiste en que el uso de los $log(\log(\frac{1}{\delta}))$ compactores sea reemplazado por un GK Sketch. Un GK Sketch, en términos sencillos, consiste en el uso de cuentas para almacenar estadísticas de los datos [19]. El algoritmo es determinístico y varía según el orden de entrada, variando así el espacio que puede ocupar esta estructura. La cota espacial consiste en $\mathcal{O}((\frac{1}{\varepsilon})\log(\log(\frac{1}{\delta})))$. Es importante mencionar que el GK sketch no soporta la unión, por lo que a veces incorporar esta modificación puede ser visto como un compromiso entre el espacio ocupado y la posibilidad de realizar la unión entre estos sketches.

Existen distintas formas de implementar el muestreo que es utilizado en el KLL, donde típicamente se usa Reservoir Sampling [8, 10] por ser una técnica no sesgada, es decir, que todos los elementos durante el procesamiento del stream tienen igual probabilidad de ser

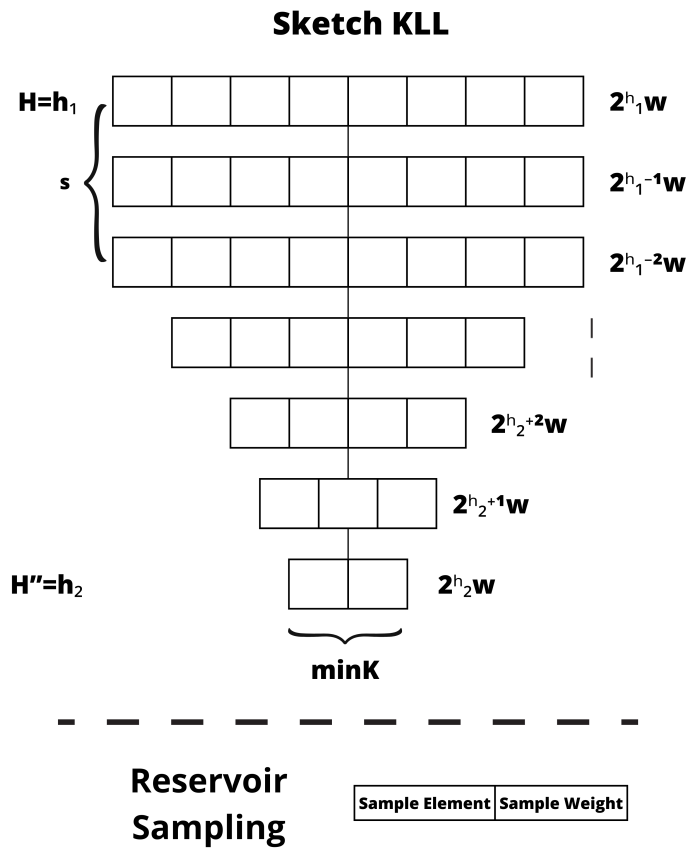


Figura 2.2.7: Ejemplo de estructura KLL, donde se utiliza Reservoir Sampling

elegidos. En el contexto del KLL, Reservoir Sampling selecciona aleatoriamente uno entre $2^{H''}$ elementos, guardando en el sketch el peso total v de los datos procesados y el elemento x_r que representa al valor seleccionado por la técnica de muestreo. Al utilizar Reservoir Sampling se diferencian tres casos al momento de ingresar en el sketch un elemento x con peso w:

1. $(v+w) \leq 2^{H''}$: Con probabilidad $\frac{w}{v+w}$ se reemplaza el valor de x_r por x. Adicionalmente se actualiza el valor de v a $v + w$.
2. $(v + w) = 2^{H''}$: Tras procesar $2^{H''}$ elementos, el sketch agrega el elemento x_r en el compactor de menor nivel, correspondiente a H''.
3. $(v + w) \geq 2^{H''}$: Con probabilidad $\frac{max(w,v)}{2^{H''}}$, el elemento de mayor peso es ingresado en el compactor de menor nivel. Además, el valor de v y x_r se actualiza a w y x cuando $v < w$. Es importante destacar que este caso solo es válido cuando $v + w < 2^{H''+1}$, debido a que en dichos casos no queda claro como incorporar el elemento muestreado en la estructura.

Capítulo 3

Desarrollo

Este trabajo contempla la implementación de los siguientes sketches de cuantiles: El MRL [18], un sketch basado en el KLL [8] y una adaptación de los sketches MRL y KLL, la cual se referirá como MRLTuple o KLLTuple. A continuación se describe en detalle la implementación de cada sketch y de las operaciones que soportan.

3.1. Implementación sketch MRL

El MRL desarrollado emplea el uso de compactores de tamaños iguales para realizar sus estimaciones, donde es necesario establecer con anterioridad la cantidad de elementos que soporte cada uno. Para ello, se proporcionan dos formas distintas de inicializar la estructura. La primera consiste en indicar la capacidad k de cada compactor, que proporciona al usuario una mayor libertad respecto al espacio y tiempo ocupado para procesar los datos ingresados. Mientras que en la segunda se proporciona la cantidad de elementos a procesar n y la cota de error ε considerada para determinar el tamaño de cada compactor. El detalle de cada forma se encuentra en los Algoritmos 1 y 2 respectivamente. Se destaca que la implementación desarrollada es dinámica, por lo que el sketch contara inicialmente con un único compactor y a medida que se necesite se incorporará el resto de ellos. La relación entre la cantidad de compactores h, la capacidad k de cada compactor y la cantidad de datos procesados n se encuentra en la ecuación 3.1.

Algoritmo 1 Inicialización del MRL proporcionando k

Input: Cantidad de elementos k que podrá almacenar un "compactor".

1: Inicializar el primer "compactor", un arreglo de tamaño k con entradas nulas.
2: Inicializar C, un arreglo que almacena "compactores"actualmente vacío.
3: Ingresar el compactor en C. Un compactor será denotado por C_j, donde j corresponde al nivel asociado del compactor.

Algoritmo 2 Inicialización del MRL proporcionando n y ε

Input: Cota de error $\varepsilon \in (0, 1)$ y el largo del stream n.

1: $k \leftarrow \varepsilon^{-1}\lceil \log(\varepsilon n) \rceil$.
2: Inicializar el primer "compactor", un arreglo de tamaño k con entradas nulas.
3: Inicializar C, un arreglo que almacena "compactores"actualmente vacío.
4: Ingresar el compactor en C. Un compactor es denotado por C_j, donde j corresponde al nivel asociado del compactor.

$$\begin{aligned} n &= k2^h \\ h &= \left\lceil \log \frac{n}{k} \right\rceil + 1 \end{aligned} \tag{3.1}$$

Una vez inicializada la estructura, es posible realizar operaciones de *insert*, *rank*, *select* y *merge*. A continuación, se explica cómo se soporta cada operación y se proporciona el pseudocódigo asociado.

3.1.1. Operación insert

Esta operación consiste en ingresar y mantener actualizada la información encontrada en el sketch. Para ello los elementos son agregados de uno en uno, donde se realizan dos pasos. Primero se ingresa un dato en el compactor de menor nivel $h = 0$, cuyos elementos tienen un peso asociado de 1. Posteriormente se verifica si es necesario efectuar una compactación producto del dato agregado, de no ser necesario se finaliza la etapa de ingreso. Por otro lado, cuando se necesita hacer una compactación se revisa que exista un compactor de nivel superior, ya que ahí se guardarán los elementos resultantes de la operación. El detalle de las operaciones de inserción y compactación se encuentran en los Algoritmos 3 y 4 respectivamente.

Se realiza un análisis amortizado para obtener la complejidad temporal de *insert*, que considera el promedio de la ejecución de n llamadas a la operación. La complejidad es determinada principalmente a partir del número de compactaciones realizadas junto al tiempo que toma efectuar dicha compactación, donde el compactor de primer nivel ejecuta $\frac{n}{2^0 k}$ compactaciones, el de segundo nivel $\frac{n}{2^1 k}$, el tercero $\frac{n}{2^2 k}$ y así sucesivamente. Dado que cada compactación toma tiempo $\mathcal{O}(k \log(k))$ por el ordenamiento de los datos, se obtiene que el análisis amortizado usando método de agregación para n operaciones está dado por:

$$\begin{aligned} &\frac{\sum_{i=0}^{\log(\frac{n}{k})} (\frac{n}{k2^i}) k \log(k)}{n} \\ &= \frac{\sum_{i=0}^{\log(\frac{n}{k})} n(\frac{1}{2^i}) \log(k)}{n} \\ &= \frac{n}{n} \sum_{i=0}^{\log(\frac{n}{k})} (\frac{1}{2^i}) \log(k) \end{aligned}$$

$$
\begin{aligned}
&= \log(k) \sum_{i=0}^{\log(\frac{n}{k})} \frac{1}{2^i} \\
&\leq \log(k) \sum_{i=0}^{\infty} \frac{1}{2^i}
\end{aligned}
$$

Utilizando suma geométrica

$$
\begin{aligned}
&= \log(k) \frac{1}{1-\frac{1}{2}} \\
&= \mathcal{O}(\log(k))
\end{aligned}
$$

Resolviendo la expresión, se obtiene que en promedio el tiempo de *insert* se encuentra acotado por $\log(k)$. Aún así, es importante aclarar que en peor caso esta operación tarda $\mathcal{O}(k \log(k) \log(\frac{n}{k}))$.

Algoritmo 3 Insert de un elemento en MRL

Input: Elemento x; arreglo de compactores C, donde C_j hace referencia al compactor encontrado en el nivel j.

1: Ingresar x en el compactor C_0.
2: **if** C_0 se encuentra lleno **then**
3: Compactación$(0, C)$.
4: **end if**

Algoritmo 4 Compactación

Input: Nivel a compactar h; arreglo de compactores C, donde C_j indica el compactor encontrado en el nivel j.

1: **if** No existe C_{h+1} **then**
2: Crear C_{h+1} con capacidad k y agregarlo a C.
3: **end if**
4: Ordenar C_h en orden creciente.
5: Seleccionar $\frac{k}{2}$ elementos en posiciones pares o impares encontrados en C_h.
6: **for** Elemento x seleccionado en C_h **do**
7: Agregar elemento x en C_{h+1}.
8: **if** C_{h+1} se llena **then**
9: Compactación$(h+1, C)$.
10: **end if**
11: **end for**
12: Vaciar C_h.

3.1.2. Operación rank

Se implementaron dos formas distintas de obtener el rank de un elemento x, estos dependen de la cantidad de elementos a consultar. Una forma consiste en determinar el rank para un único elemento. En este caso, se comparan los valores encontrados en los compactores de la estructura. Cuando un valor comparado es menor o igual a x se agrega el peso del compactor asociado (2^h) a la estimación del rank. Cuando se termine de comparar el último compactor se tendrá la estimación del rank del elemento consultado. La otra forma consiste en estimar el rank de varios elementos a la vez, para lograrlo se debe mantener un orden de los elementos revisados, donde se organizan los datos vistos de todos los compactores del MRL en un arreglo con formato (elemento, frecuencia). Una vez se tengan los datos ordenados se puede estimar el rank de cada elemento sumando todas las frecuencias asociadas a aquellos elementos cuyo valor sea menor o igual al indicado. Estas formas son implementadas en los Algoritmos 5 y 6.

Es importante destacar que la operación rank se implementa de dos formas distintas, porque cuando se quiere consultar por muchos elementos al mismo tiempo resulta ineficiente ir obteniendo el rank de cada elemento individualmente. La consulta individual tiene una complejidad temporal lineal respecto a la cantidad de elementos encontrados en el sketch correspondiente a $\mathcal{O}(kH)$, donde k es el tamaño de cada compactor y H el número de compactores. Por su parte, la consulta grupal requiere de ordenar el arreglo generado por los elementos del sketch, obteniendo así una complejidad temporal de $\mathcal{O}(kH \log kH)$ para realizar M consultas simultáneas. De lo anterior, se recomienda emplear rank grupal cuando se cumpla que $\log kH < M$, ya que el costo asociado a ordenar todos los elementos del sketch es amortizado por la cantidad de consultas realizadas.

Algoritmo 5 Rank Individual MRL

Input: Elemento x del cual se quiere determinar el rank; nivel más alto del MRL H; arreglo de compactores C, donde C_j hace referencia al compactor encontrado en el nivel j.
Output: Rank r asociado al elemento x.

```
1: Rank ← 0.
2: for h = 0 hasta el número total de compactores H do
3:     for Elemento m en C_h do
4:         if m ≤ x then
5:             Rank += 2^h.
6:         end if
7:     end for
8: end for
9: return Rank.
```

Algoritmo 6 Rank Grupal en MRL

Input: Grupo de Elementos X, de los cuales se quiere determinar sus ranks; nivel más alto del MRL H; arreglo de compactores C, donde C_j hace referencia al compactor encontrado en el nivel j.

Output: Arreglo con los ranks asociado a cada uno de los elementos en X.

1: $R \leftarrow$ Arreglo de ranks inicialmente vacío.
2: $M \leftarrow$ Arreglo de pares con formato {elemento, frecuencia estimada}.
3: **for** $h = 0$ hasta H **do**
4: **for** Elemento m en C_h **do**
5: Insertar el par {elemento m, frecuencia 2^h} a M.
6: **end for**
7: **end for**
8: Ordenar M de manera ascendente según los elementos.
9: **for** Elemento x en X **do**
10: rank $\leftarrow$ Suma de las frecuencias en M de todos los elementos menores o iguales a x
11: Agregar rank al arreglo R.
12: **end for**
13: Return R, el cual corresponde a un arreglo con los ranks asociados a los elementos proporcionados en X respectivamente.

3.1.3. Operación select

Esta operación busca obtener el elemento cuyo rank sea el proporcionado y el Algoritmo 7 lo describe. Para empezar se traspasan los elementos encontrados en los compactores a un arreglo con formato (elemento, frecuencia), el cual es posteriormente ordenado por el valor correspondiente a sus elementos. Luego, se obtiene el índice del primer elemento en el arreglo cuya suma de frecuencias hasta él sea mayor o igual al rank consultado. Finalmente, se entrega el elemento del arreglo asociado al índice calculado. Al igual que rank grupal, esta operación toma en peor caso tiempo $\mathcal{O}(kH \log(kH))$. Cabe mencionar que el arreglo de pares (elemento, frecuencia) puede ser almacenado para utilizarse en futuras consultas dentro de un período de tiempo establecido por el usuario. Dentro de dicho período se reduce el tiempo ocupado por las consultas select y rank, debido a que se amortiza el costo asociado a la construcción del arreglo y se realiza búsqueda binaria sobre este.

Adicionalmente, se puede soportar la operación de cuantil(q) producto de incorporar la operación select. Para ello, es necesario conocer la cantidad de elementos ingresados en la estructura n. En tal caso, se puede implementar esta operación como select(qn) donde q, un valor entre $[0, 1]$, corresponde a la proporción de elementos que serán divididos por el valor obtenido. Esto se encuentra reflejado en el Algoritmo 8.

Algoritmo 7 Select en MRL y KLL

Input: El rank r asociado al elemento consultado; nivel del compactor de mayor nivel H; arreglo de compactores C, donde C_j hace referencia al compactor encontrado en el nivel j.

Output: Elemento x cuyo rank corresponda a r.

1: $M \leftarrow$ Arreglo de pares con formato {elemento, frecuencia}.
2: **for** $h = 0$ hasta H **do**
3: **for** Elemento m en C_h **do**
4: Insertar el par {elemento m, frecuencia 2^h} a M.
5: **end for**
6: **end for**
7: Ordenar M de manera ascendente según los elementos.
8: $i \leftarrow$ Menor índice en M tal que $\sum_{j=0}^{i} M[j].second \geq r$.
9: Return $M[i].first$.

Algoritmo 8 Cuantil en KLL y MRL

Input: Cuantil de orden q a consultar; número de elementos procesados hasta el momento n.

Output: Elemento x asociado al cuantil de orden q.

1: Return Select(qn).

3.1.4. Operación merge

La unión entre dos sketches MRL es posible mediante el traspaso de los elementos desde una estructura a otra. En este caso, los datos son enviados desde la estructura que tiene menos compactores a la que tiene más. El proceso parte desde el nivel mas bajo, es decir $h = 0$, y consiste en ingresar los elementos encontrados en un compactor a otro asociado al mismo nivel. A veces este proceso puede necesitar la ejecución de varias compactaciones, en dichos casos, se dará prioridad a finalizar la compactación antes de seguir ingresando más elementos desde un compactor a otro. La complejidad temporal de esta operación es equivalente al análisis de peor caso explicado en *Insert*, el cual corresponde a $\mathcal{O}(H_{menor} K_{menor} \log (H_{menor} K_{menor}))$.

Algoritmo 9 Merge en MRL

Input: Dos MRL M_1 y M_2; la altura correspondiente a sus niveles más altos H_1 y H_2; arreglo de compactores de cada MRL C_1 y C_2.

Output: MRL M que une los elementos encontrados en los MRL ingresados.

```
1: M_i ← MRL que tiene mayor altura.
2: M_j ← MRL que tiene menor altura.
3: for h = 0 hasta H_j do
4:     for Elemento m en C_jh do
5:         Ingresar m en C_ih.
6:         if C_ih se encuentra lleno then
7:             Compactación(ih, C).
8:         end if
9:     end for
10: end for
```

3.2. Implementación del sketch KLL

En esta implementación del KLL se utilizan varios parámetros para determinar el largo de cada compactor, sus niveles asociados y la tasa de muestreo. Se proporcionan dos alternativas para inicializar la estructura. La primera consiste en proporcionar parámetros que permiten establecer la forma de la estructura, estos son la cota de error aceptada ε, la probabilidad de que no se cumpla dicha cota de error δ, el factor c empleado para ir disminuyendo la capacidad de cada compactor exponencialmente, la capacidad mínima de cada compactor $minK$ y una estimación de la cantidad de elementos que serán procesados. La otra alternativa se basa en determinar manualmente la forma de los compactores y sus niveles asociados. La determinación de la cantidad de compactores y sus capacidades asociadas se encuentra en los Algoritmos 10 y 11.

Este sketch soporta las operaciones de Insert, Rank, Select y Merge. De estas, Rank y Select siguen el mismo procedimiento que en el MRL, encontrados en los Algoritmos 5, 6 y 7. Se destaca que la complejidad temporal de estas operaciones cambia respecto a las mencionadas en MRL, puesto que la cantidad de elementos almacenados en el sketch dependen de la capacidad del compactor de mayor nivel k_H y del factor de reducción c utilizado. Dado que la capacidad en cada nivel disminuye exponencialmente, se tiene que la cantidad total de elementos desde el compactor de nivel H'' hasta H corresponden a:

$$\begin{aligned} &\sum_{i=H''}^{H} k_H c^{H-i} \\ \leq &\sum_{i=0}^{H} k_H c^i \end{aligned}$$

$$\leq \sum_{i=0}^{\infty} k_H c^i$$
$$= k_H \sum_{i=0}^{\infty} c^i$$

Dado que $c < 1$ y al Utilizar suma geométrica

$$= k_H \frac{1}{1-c}$$

Obteniendo así una complejidad temporal de $\mathcal{O}(k_H)$ para la consulta Rank individual y $\mathcal{O}(k_H \log(k_H))$ para Rank grupal y Select. Por su parte, las operaciones Insert y Merge necesitan modificaciones producto de incorporar la técnica de muestreo, las cuales se encuentran descritas en las siguientes secciones.

Algoritmo 10 Inicialización de KLL

Input: Cota de error ε; probabilidad que la cota de error no se cumpla δ; factor de disminución c; capacidad mínima $minK$; y la estimación de la cantidad de elementos a procesar n.

1: Nivel del compactor de mayor nivel $H \leftarrow \log(\varepsilon n)$.
2: Capacidad del compactor de mayor nivel $k_H \leftarrow \frac{1}{\varepsilon} \log(\log(\frac{1}{\delta}))$.
3: Número de compactores de mayor nivel que tienen misma capacidad $s \leftarrow \log(\log(\frac{1}{\delta}))$.
4: Nivel del compactor de menor nivel $H'' \leftarrow H - \lceil \frac{\log k_H}{\log c} \rceil$.
5: Variables utilizadas por Reservoir Sampling: elemento $x_r \leftarrow$ nulo, y peso asociado $w \leftarrow 0$.
6: Tasa de muestreo del Reservoir Sampling $\leftarrow 2^{H''}$.
7: Inicializar C, un arreglo que almacena "compactores"actualmente vacío.
8: **for** $h = H''$ hasta H **do**
9: **if** $h > (H - s)$ **then**
10: $k_h \leftarrow max(minK, c^{H-h} k_H)$.
11: **else**
12: $k_h \leftarrow k_H$.
13: **end if**
14: $C_h \leftarrow$ Arreglo vacío de tamaño k_h, que corresponderá al compactor de nivel h.
15: Ingresar C_h en C.
16: **end for**

Algoritmo 11 Inicialización manual del KLL

Input: Nivel del compactor de mayor nivel H; arreglo de compactores A.

1: Variables utilizadas por Reservoir Sampling elemento $x_r \leftarrow$ nulo, y peso asociado $w \leftarrow 0$.
2: Tasa de muestreo del Reservoir Sampling $\leftarrow 2^{H''}$.
3: $H'' \leftarrow H - A.size() + 1$.
4: Inicializar C, un arreglo que almacena "compactores"actualmente vacío.
5: Vaciar todos los compactores de A e ingresarlos en C.

3.2.1. Operación insert

Los datos recibidos se muestrean mediante Reservoir Sampling, que indica mediante un booleano cuándo se debe añadir un elemento en el compactor de menor nivel. Cuando este booleano es afirmativo se agrega el elemento seleccionado x_r en el compactor de menor nivel, donde se verifica si se requiere realizar una compactación. Los pseudocódigos asociados se encuentran en los Algoritmos 12 y 13.

Se realiza un análisis amortizado usando método de agregación para determinar la complejidad de la operación *insert*. En esta operación se considera el tiempo ocupado en realizar Reservoir Sampling, la inserción de los elementos en los compactores y sus respectivas compactaciones cuando sea necesario. Reservoir Sampling procesa los n elementos, donde cada uno demora tiempo constante en ser muestreado. La inserción de un elemento en un compactor también toma tiempo constante, por lo que la complejidad de la operación insert proviene de las compactaciones. Cada compactor ocupa tiempo $\mathcal{O}(k_h \log(k_h))$ en compactar el nivel h, por lo que basta con determinar la cantidad de compactaciones que realiza cada compactor para obtener la cota buscada. El compactor de menor nivel, correspondiente al nivel H'', procesa $\frac{n}{2^{H''}}$ elementos y al tener una capacidad asociada de $k_{H''}$ se compacta $\frac{n}{2^{H''}k_{H''}}$ veces. El siguiente compactor, cuyo nivel corresponde a $H''+1$, procesa la mitad de los datos que el compactor anterior, es decir $\frac{n}{2^{H''+1}}$ elementos, y al tener una capacidad asociada de $k_{H''+1}$ se compacta $\frac{n}{2^{H''+1}k_{H''+1}}$ veces, esto se puede extender para todos los siguientes compactores. Dado que se conoce la capacidad del compactor de mayor nivel k_H y el factor de reducción c empleado, la capacidad del resto de los compactores se determina como $k_h \approx k_H(c^{H-h})$. Considerando lo anterior y que los elementos añadidos por Reservoir Sampling son agregados al compactor de nivel H'', es que el promedio del tiempo total de n operaciones está dado por:

$$\begin{aligned}
&\frac{\sum_{i=H''}^{H} \frac{n}{2^{i-H''}k_i} k_i \log(k_i)}{n} \\
&= \frac{\sum_{i=H''}^{H} \frac{2^{H''}n}{2^i} \log(k_i)}{n} \\
&= \frac{2^{H''}n}{n} \sum_{i=H''}^{H} \frac{1}{2^i} \log(k_i)
\end{aligned}$$

Debido a que $k_h = k_H c^{H-h}$.

$$
\begin{aligned}
&= 2^{H''} \sum_{i=H''}^{H} \frac{1}{2^i} \log\left(k_H c^{H-i}\right) \\
&= 2^{H''} \sum_{i=H''}^{H} \frac{1}{2^i} (\log\left(k_H\right) + (H-i)\log\left(c\right)) \\
&= 2^{H''} \sum_{i=0}^{H-H''} \frac{1}{2^{i+H''}} (\log\left(k_H\right) + (H-(i+H''))\log\left(c\right)) \\
&= \sum_{i=0}^{H-H''} \frac{\log\left(k_H\right) + (H-H'')\log\left(c\right)}{2^i} - \log\left(c\right) \sum_{i=0}^{H-H''} \frac{i}{2^i} \\
&\leq \sum_{i=0}^{H-H''} \frac{\log\left(k_H\right) + (H-H'')\log\left(c\right)}{2^i} - \log\left(c\right) \\
&= (\log\left(k_H\right) + (H-H'')\log\left(c\right)) \sum_{i=0}^{H-H''} \frac{1}{2^i} - \log\left(c\right) \\
&\leq (\log\left(k_H\right) + (H-H'')\log\left(c\right)) \sum_{i=0}^{\infty} \frac{1}{2^i} - \log\left(c\right) \\
&= (\log\left(k_H\right) + (H-H'')\log\left(c\right)) \frac{1}{1-\frac{1}{2}} - \log\left(c\right) \\
&= 2\log\left(k_H\right) + 2\log\left(c\right)(H-H'') - \log\left(c\right) \\
&= 2\log\left(k_H\right) + 2\log\left(c\right)(H-(H-\left\lceil \frac{\log k_H}{\log c} \right\rceil)) - \log\left(c\right) \\
&= 2\log\left(k_H\right) + 2\log\left(k_H\right) - \log\left(c\right)
\end{aligned}
$$

A partir de la ecuación anterior y que los valores k_H y $\log(c)$ son constantes, se obtiene que en promedio el tiempo ocupado por la operación *insert* es lineal, es decir $\mathcal{O}(1)$.

Algoritmo 12 Insert KLL

Input: Elemento a ingresar x; peso asociado del elemento ingresado w; arreglo de compactores C; y el menor nivel del KLL H''.

```
1: if Sample(x, w) then
2:     Ingresar x en el compactor C_{H''}.
3:     if C_{H''} se llenó then
4:         Compactación(H'', C).
5:     end if
6: end if
```

Algoritmo 13 Reservoir Sampling KLL

Input: Elemento a ingresar x; peso asociado al elemento ingresado w; el menor nivel del KLL H''; elemento seleccionado hasta el momento por Reservoir Sampling x_r; y el peso asociado al muestreo v.

Output: Booleano que indica si se debe ingresar el elemento muestreado en un compactor.

elementoEsSeleccionado $\leftarrow$ false.
if $v + w > 2^{H''}$ **then**
    Con probabilidad $\frac{max(w,v)}{2^{H''}}$ el elemento de mayor peso se ingresa en el compactor $C_{H''}$.
    $x_r \leftarrow$ elemento de menor peso entre x_r y x.
    $v \leftarrow min(v, w)$.
    Return elementoEsSeleccionado.
end if
if $v + w \leq 2^{H''}$ **then**
    Con probabilidad $\frac{w}{v+w}$ se reemplaza x_r por x.
end if
if $v + w = 2^{H''}$ **then**
    elementoEsSeleccionado $\leftarrow$ true.
end if
$v \leftarrow (v + w)$módulo$2^{H''}$.
Return elementoEsSeleccionado.

3.2.2. Operación merge

Para implementar la unión entre dos KLL se ingresan los elementos encontrados desde una estructura a otra. Sin embargo, hay que considerar el nivel de cada compactor y la tasa de muestreo aplicada. Los datos son enviados desde un KLL que se denomina K_j a otro que se denota K_i. Para determinar qué estructura recibe los datos se compara cuál es la que tiene un compactor de mayor nivel.

El proceso de unión comienza ingresando los datos provenientes del muestreo de K_j a K_i. Para ello se compara el valor de la suma de los pesos encontrados en el muestreo de cada sketch, donde se realiza un serie de pasos distintos dependiendo si el valor de la suma supera el límite soportado por K_i. De no ser superado, el elemento x_r y su peso asociado v provenientes de K_j son ingresados en K_i mediante el método Insert. Por otro lado, cuando se supera el límite se opta por ingresar el elemento entre uno de dos compactores en K_i dependiendo del peso proveniente del elemento a ingresar, estos compactores deben tener un peso asociado mayor y otro menor, cuyos niveles denotaremos por h_R y h_r respectivamente. Para seleccionar el compactor se calcula una probabilidad de agregar el elemento en el compactor de nivel h_R, que considera los pesos de los compactores y el peso del elemento a ingresar. En caso que no se cumpla dicha probabilidad el elemento es agregado en h_r. Además, es importante destacar que esta probabilidad aumenta mientras mayor sea el peso del elemento.

La unión continúa trasladando todos los elementos encontrados en los compactores provenientes de K_j. Cuando existe un compactor en K_i que tenga los mismos pesos o nivel que

aquel en K_j, los datos son ingresados desde un compactor a otro. Si no se cumple, los elementos son agregados en K_i mediante el método Insert, indicando el peso del compactor asociado.

La complejidad temporal de la unión o merge considera el traspaso de la información contenida tanto en Reservoir Sampling como en los compactores, sin embargo, incorporar los datos del muestreo desde un sketch a otro es realizado en tiempo constante. Por otra parte, la complejidad temporal se encuentra acotada cuando se agregan los elementos desde un compactor a otro, donde el peor caso involucra ordenar todos los compactores producto de la compactación. Considerando que las constantes indicadas hacen referencia al sketch K_i, que tiene el compactor de mayor nivel, tenemos que la complejidad se encuentra acotada por $\mathcal{O}(H - H'')(k_H \log k_H)$.

Algoritmo 14 Merge KLL

Input: Dos KLL K_1 y K_2; la altura correspondiente a sus niveles más altos H_1 y H_2; arreglo de compactores de cada KLL C_1 y C_2, donde C_{h1} hace referencia al compactor de nivel h en el KLL K_1; los elementos asociados al muestreo de cada KLL y sus pesos asociados x_{r1}, v_1 y x_{r2}, v_2.

1: $K_i \leftarrow$ KLL que contiene el compactor de mayor nivel.
2: $K_j \leftarrow$ KLL que no contiene el compactor de mayor nivel.
3: **if** $v_i + v_j < 2^{H_i''+1}$ **then**
4: Ingresar los datos del muestreo de K_j en K_i mediante K_i.Insert(x_{rj}, v_j)
5: **else**
6: $h_r \leftarrow \lfloor log(v_j) \rfloor$.
7: $h_R \leftarrow \lceil log(v_j) \rceil$.
8: Con probabilidad $\frac{v_j - 2^{h_r}}{2^{h_R} - 2^{h_r}}$ se ingresa x_{rj} en el compactor de nivel h_R. De lo contrario se ingresa en el compactor de nivel h_r.
9: **end if**
10: **for** $h = H_j''$ hasta H_j **do**
11: **for** Elemento m en C_{hj} **do**
12: **if** C_{hi} existe **then**
13: Ingresar m en C_{hi}.
14: **if** C_{hi} se encuentra lleno **then**
15: Compactación(hi, C).
16: **end if**
17: **else**
18: Ingresar m mediante K_i.Insert$(m, 2^h)$.
19: **end if**
20: **end for**
21: **end for**

3.3. Implementación del sketch KLLTuple

En esta sección se describe la implementación del sketch KLLTuple, que consiste en una adaptación del KLL implementado para soportar consultas para una aplicación de monitoreo de red. Esta aplicación consiste en estimar los flujos más significativos que se encuentran dentro de los top-k flujos con mayores payloads, recordando que un payload hace referencia al tamaño de los datos en un paquete de red que no es empleado para su transporte. Medir el tráfico existente en una red es importante, ya que permite detectar actividades anormales existentes, afectando la seguridad de una red. Según la medición de un tráfico de red, el personal puede implementar y gestionar contramedidas cuando se detecten actividades anómalas [4].

Los sketches descritos anteriormente (MRL y KLL) solo permiten estimar estadísticas relacionadas con la distribución de los flujos y no permiten incluir información que sea de interés de cada flujo. Para abarcar esta alternativa, se explora la posibilidad de incluir más información en el sketch KLL con el fin de soportar los datos asociados a los payloads de los flujos. Con dicha información, cada compactor tiene que ser modificado para soportar una tupla que contenga datos respecto al identificador del flujo y el payload asociado. Este sketch que soporta tuplas será denominado posteriormente como KLLTuple o MRLTuple, dependiendo del tipo de sketch en el que se base.

Cada compactor necesita establecer cómo ordenar sus elementos, pues en distintas operaciones como la *Compactación* y *Select* es clave mantener organizada la información para poder realizar las estimaciones con un mínimo error relacionado. El orden de los datos se determinó según el payload asociado a cada flujo, pues no es relevante realizar estimaciones respecto a los identificadores de los flujos. Aún así, el identificador aporta información importante para poder procesar los datos. Un ejemplo de lo anterior ocurre previo a la compactación, donde se incorpora una operación denominada *reducción* que busca elementos con los mismos identificadores de flujos y los une sumando los payloads de los elementos. El objetivo de esta operación consiste en reducir el número de veces que se realiza la compactación y, a su vez, la cantidad de veces que se puede agregar error en las estimaciones realizadas.

Un factor importante en el sketch es que los flujos son eliminados durante la compactación, esto provoca que no se puedan realizar consultas respecto a flujos específicos. Sin embargo, se incorpora el uso de un min-heap para poder mantener y responder una consulta respecto a aquellos flujos con mayor payload, la cual se denomina *TopFlows*. El min-heap implementado en este trabajo consiste en una estructura de datos que organiza una cantidad limitada de elementos en base a cuál contiene un menor valor, este valor es representado mediante el payload asociado a un flujo y su identificador respectivo. Dado que el min-heap cuenta con capacidad limitada, cada vez que un nuevo elemento es agregado se verifica si existe algún flujo con el mismo identificador, en dicho caso se actualiza el valor del payload asociado al elemento en la estructura. En caso contrario, se comprueba si la estructura se encuentra llena o no. En el primer caso se compara el valor de los payloads del elemento a insertar y del flujo con menor payload en el min-heap, si el elemento a ingresar contiene mayor payload reemplaza al flujo con menor payload, mientras que si tiene menor payload no es incorporado

en la estructura. En el segundo caso el elemento simplemente es incorporado dentro de la estructura. Cabe destacar que el min-heap no modifica de gran manera el funcionamiento del sketch, ya que únicamente recibe los datos que son eliminados durante la compactación.

Para inicializar un sketch con tuplas es necesario que el usuario indique cuántos elementos tendrá su min-heap. Para ello se debe proporcionar un parámetro adicional al momento de generar un sketch KLLTuple o MRLTuple, el cual corresponde al número máximo de elementos t que pueden ser almacenados dentro del min-heap. Por otra parte, la única diferencia en la inicialización entre los sketches MRL y KLL en comparación con sus variantes con tuplas consiste en el tipo de elemento con el que trabajan.

A continuación se describe en detalle respecto a las modificaciones realizadas sobre los sketches MRL o KLL para que las operaciones *Insert, Rank, Select y Merge* soporten el uso de tuplas. Adicionalmente, los cambios realizados en *Rank y Select* no afectan las cotas temporales obtenidas respecto a los sketches sobre los que están basados, sean MRLTuple o KLLTuple.

3.3.1. Operación insert

El procedimiento para ingresar los datos en los sketches con tuplas es similar a como se realiza en el MRL y KLL. La diferencia radica en que se incorpora una operación denominada *Reducción* posterior a cuando se llena un compactor y previo a su compactación. Esta operación ordena los elementos a partir de sus identificadores de flujo, uniendo los pares de elementos con mismo identificador de flujo. En caso que la reducción no ejecute ninguna unión en el compactor, se continua con la compactación. En caso contrario la reducción finaliza el proceso de inserción, debido a que el compactor libera espacio que puede ser usado posteriormente.

La compactación recibe algunas modificaciones respecto a su implementación en el MRL y KLL, puesto que los elementos son ordenados a partir de sus payloads y los elementos que no son seleccionados durante esta etapa ahora son ingresados en un min-heap mediante una operación denominada *InsertHeap*. Este min-heap se encarga de mantener a los flujos con mayor payload considerando un número limitado de elementos. Estas modificaciones se encuentran presentes en los Algoritmos 15, 16, 17 y 18.

Es importante notar que el Algoritmo 15 contiene los pasos para hacer insert en un MRLTuple. Sin embargo, basta con incorporar la etapa de muestreo para que el algoritmo funcione en el contexto de un KLLTuple.

Algoritmo 15 Insert en MRLTuple

Input: Elemento x, compuesto por el identificador de flujo x_f y su payload correspondiente x_p; arreglo de compactores C, donde C_j hace referencia al compactor encontrado en el nivel j.

```
1: Ingresar x en el compactor C_0.
2: if C_0 se encuentra lleno then
3:     Reducción(0, C)
4:     if C_0 sigue lleno then
5:         Compactación(0, C).
6:     end if
7: end if
```

Algoritmo 16 Reducción de un compactor en un sketch con tuplas

Input: Nivel del compactor a reducir h; arreglo de compactores C, donde C_j indica al compactor encontrado en el nivel j.

```
1: Ordenar C_h a partir del identificador de los flujos.
2: for Elemento m en C_h do
3:     if Se encuentra un elemento x en C_h con el mismo identificador de flujo que m then
4:         Payload de m ← Payload de m + Payload de x.
5:         Eliminar x.
6:     end if
7: end for
```

Algoritmo 17 Compactación para un sketch con tuplas

Input: Nivel a compactar h; arreglo de compactores C, donde C_j hace referencia al compactor encontrado en el nivel j; min-heap asociado a la estructura.

```
1: if No existe C_{h+1} then
2:     Crear C_{h+1} con capacidad k.
3: end if
4: Ordenar C_h a partir de los payloads de los elementos en orden creciente.
5: Seleccionar k/2 elementos en posiciones pares o impares encontrados en C_h.
6: Los elementos seleccionados son agregados en C_{h+1}.
7: for Elemento m que no fue seleccionado do
8:     Ingresar m en el min-heap.
9: end for
10: if C_{h+1} se llena then
11:     Compactación(h + 1, C).
12: end if
13: Vaciar C_h.
```

Algoritmo 18 Insert en el min-heap de un sketch con tuplas

Input: Elemento x, donde x_f se refiere al identificador de flujo del elemento x y x_p se refiere al payload del elemento x; min-heap del sketch; cantidad máxima de elementos que puede soportar el heap t.

```
1: mx ← elemento con menor payload en el min-heap.
2: if Existe un elemento m en el min-heap con identificador x_f then
3:     Actualizar el valor del payload de m a la suma entre los payloads de x y m.
4:     Mantener el orden del heap mediante operaciones heapify[a].
5: else
6:     if min-heap tiene t elementos then
7:         if mx_p > x_p then
8:             Reemplazar mx por x.
9:             Mantener el orden del heap mediante operaciones heapify.
10:        end if
11:    else
12:        Ingresar mx en el heap.
13:        Mantener el orden del heap mediante operaciones heapify.
14:    end if
15: end if
```

[a]heapify: Operación del min-heap que mueve los elementos guardados para mantener el orden dentro de la estructura

3.3.2. Rank, Select y Merge

Las modificaciones realizadas en las operaciones de *Rank* y *Select* consisten en que ahora responden a consultas respecto a los payloads, proporcionando estimaciones de la distribución de los payloads encontrados en la red. Ambas operaciones deben ordenar y organizar su información considerando solo el componente de los payloads. Asimismo, la unión entre dos estructuras que soportan tuplas solo debe tener en cuenta que los elementos a traspasar desde un sketch a otro consisten de tuplas, notando que cuando un compactor se llena se ejecutan los procedimientos de reducción y compactación encontrados en los Algoritmos 16 y 17 respectivamente.

3.3.3. TopFlows

El objetivo de *TopFlows* radica en estimar los flujos con mayores payloads y el valor de sus payloads asociados. Para ello se considera el comportamiento del min-heap empleado, pues se espera que la estructura mantenga una determinada cantidad de flujos con mayor payload luego de su eliminación durante la etapa de compactación.

El Algoritmo 19 refleja que la operación *TopFlows* únicamente extrae los elementos del min-heap que contienen las estimaciones realizadas por el sketch. Sin embargo, la estimación puede incorporar los valores almacenados en los compactores del sketch. Para ello se suman

los payloads de aquellos elementos con los mismos identificadores de flujo encontrados en el min-heap. Esto último involucra ejecutar una búsqueda adicional que requiere un tiempo de cómputo significativo y, en la mayoría de los casos, tiene un impacto despreciable en el valor del payload de las estimaciones obtenidas.

Algoritmo 19 Algoritmo TopFlows para un sketch con tuplas

Input: min-heap asociado al sketch.

1: Crear un arreglo R de elementos con formato (flujo, payload) actualmente vacíos.
2: Extraer los elementos del min-heap e ingresarlos en R.
3: Return R.

3.3.4. Complejidad Temporal

Incorporar el uso de un min-heap afecta la complejidad temporal de las operaciones *Insert* y *Merge*, puesto que cada vez que se ejecuta una compactación se debe buscar si los elementos no seleccionados contienen un mismo identificador de flujo que alguno ya existente en el heap. Los Algoritmos asociados a la compactación y el ingreso de elementos en el heap se encuentran en 17 y 18 respectivamente. Como los elementos son agregados de uno en uno en el heap, la complejidad temporal para la compactación en un sketch con tuplas está dada por $\mathcal{O}(k \log(k) + \frac{k*t}{2})$, donde t hace referencia a la cantidad de elementos soportados por el min-heap.

Capítulo 4

Evaluación Experimental

4.1. Entorno de evaluación

4.1.1. Implementación

La implementación de las estructuras desarrolladas en esta memoria de título se encuentra en el repositorio de GitHub disponible en `https://github.com/Landerer0/MT`.
Los experimentos fueron desarrollados en el servidor Chome de la Universidad de Concepción, utilizando el compilador g++ para el lenguaje de programación C++. La ejecución de dichos programas en el servidor contaba con un equipo de 80 procesadores Intel(R) Xeon(R) Gold 5320T a 2.30 GHz, y con una memoria principal utilizable de aproximadamente 295 GB.

4.1.2. Datasets

El análisis de cuantiles respecto a direcciones IPs de origen o destino puede ser empleado, por ejemplo, para estimar el porcentaje de paquetes de red que provienen desde un rango IPs especificado, este porcentaje se calcula como la diferencia entre los ranks de la IP de mayor y menor valor consultado. De lo anterior, se utilizan datasets que contienen datos respecto a trazas de red provenientes de *Mendeley*, *CAIDA* y *Mawilab* [20, 21, 22]. Estos datasets se dividen en dos tipos. El primero contiene únicamente la IP origen de los paquetes de red, los cuales se usan para realizar experimentos en las operaciones *Rank* y *Select* del KLL y MRL. La tabla 4.1.1 indica el nombre de cada uno de los datasets usados de este tipo, junto al promedio μ y a la desviación estándar σ de las IPs, el número total de IPs M y el número de IPs distintas N. Adicionalmente, se proporciona el histograma de algunos datasets en la Figura 4.1.1, la cual indica la frecuencia de distintos paquetes de red ordenados de manera descendente según el valor de dicha frecuencia. El segundo tipo de datasets contiene datos usados por los sketches KLLTuple y MRLTuple, que indican la IP fuente del flujo perteneciente a cada paquete de red y su respectivo payload. La tabla 4.1.2 indica la cantidad de flujos distintos N, además

Traza	M	N	μ_{IP}	σ_{IP}
Chicago-20080319	3938619	167768	2143376436	1251815849
Chicago-20080515	12242152	199412	2167676793	1268638336
Chicago-20110608	27046414	393237	2047032563	1224947404
Chicago-20150219	15962528	252239	2171969018	1442172853
Chicago-20160121	31197995	135269	1967932310	1302981972
Mawi-20161201	94738984	5083756	2117818433	1187492119
Mawi-20171101	115486661	4145741	2047350247	1247074471
Mawi-20181201	76066888	4674492	1848456630	1275616854
Mawi-20191102	62478145	4633485	2466169859	1339168259
Mawi-20200901	119923870	5394529	1562225665	1249521137
Mendeley	2668026	101833	3025098862	601664481

Tabla 4.1.1: Estadísticas respecto a las trazas de red en distintos datasets para realizar experimentos con los sketches KLL y MRL.

Traza	N	μ_p	σ_p
flujosChicago-2016	135051	232519	5445408
flujosMawi-2016	5083739	12921	5285087

Tabla 4.1.2: Estadísticas respecto a las trazas de red en distintos datasets para realizar experimentos en KLLTuple.

de la media μ_p y desviación estándar σ_p de los payloads encontrados en dichos flujos para algunos datasets de este tipo.

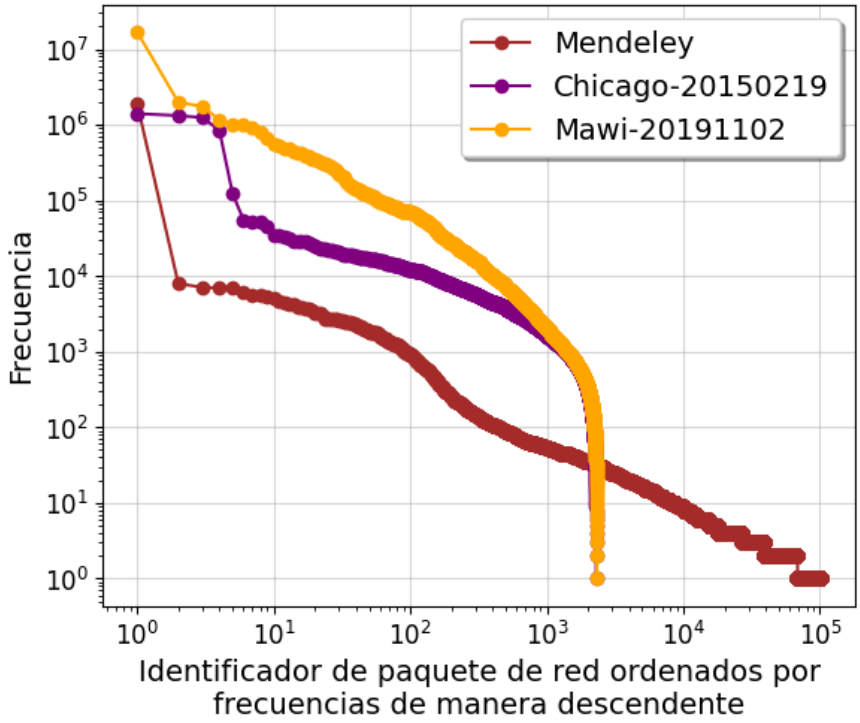


Figura 4.1.1: Histogramas que indica las frecuencias asociados a distintos paquetes de red ordenados por sus frecuencias en orden descendente encontrados en los datasets Chicago-20150219, Mendeley y Mawi-20191102.

4.1.3. KLL ApacheDatasketch

Existe una biblioteca desarrollada por la comunidad denominada ApacheDatasketch [23] que contiene diversos tipos de sketches para resolver distintos problemas. Uno de estos sketches consiste en el KLL sketch, el cual soporta llamadas a las operaciones *Rank* y *Select*, que permite usarlo para comparar el KLL y MRL implementado. El KLL de ApacheDatasketch no realiza una tasa de muestreo interna, sino que procesa todos los datos y estos son almacenados inicialmente en el compactor de primer nivel de dicha estructura, el cual contiene una capacidad indicada por parámetro denominada k_p. Una vez llenado un compactor, se calcula la capacidad del siguiente compactor como:

$$k_{\text{nivel}} = k_p \frac{2}{3}^{nivel-1} \tag{4.1}$$

Es importante destacar que este KLL emplea espacio dinámico, por lo que cada vez que sea necesario añadir un nuevo compactor se le agregará espacio a la estructura, el cual no será necesariamente ocupado por el nuevo compactor. Para aprovechar la mayor cantidad de espacio posible, el sketch organiza la información en dos arreglos que funcionan en conjunto. El arreglo *items* es el encargado de almacenar todos los elementos asociados a los compactores, mientras que el arreglo *lvl* mantiene un registro que indica el índice donde comienza

k_p	ε
300	0.009
340	0.008
390	0.007
450	0.006
550	0.005
700	0.004
900	0.003
1300	0.002

Tabla 4.1.3: Relación entre valor de k_p y ε para el KLL de ApacheDatasketch.

y termina cada nivel en el arreglo *items*. La particularidad del sketch proviene en que todos los elementos asociados a un compactor mantienen información útil para la estructura, la excepción ocurre en el compactor de primer nivel, donde se necesita espacio para almacenar los elementos que son ingresados. Durante una compactación la mitad de los elementos son agregados al compactor de mayor nivel, esto se realiza seleccionando los elementos a mantener y unirlos mediante merge sort. Posteriormente el espacio asociado a los elementos que no son seleccionados se incorporan al compactor de primer nivel actualizando el arreglo *items* y *lvl* acordemente.

Para poder comparar esta estructura con el KLLTuple y MRLTuple, se realizaron ciertas modificaciones a la compactación de la estructura. Estas modificaciones son:

- Se realiza una reducción previo a una compactación, por lo que los datos son ordenados respecto a los identificadores de los flujos. Existen dos casos dependiendo de si algún elemento es reducido o no. En caso afirmativo, el espacio liberado por la unión de elementos es redireccionado al compactor de primer nivel y no se procede con la compactación, en caso contrario se continúa con dicha operación. Independiente del caso es importante mantener ordenados los datos para no interrumpir el merge sort de otro compactor, de manera que se vuelve a realizar un ordenamiento de los datos respecto a su payload.
- Los elementos que no son seleccionados durante la compactación son enviados a un min-heap, el cual funciona de la misma manera que en el KLL y MRL implementados. La cantidad de espacio ocupado por dicha estructura es indicada al inicializar el KLL.

Debido a que el KLL de ApacheDatasketch es inicializado mediante el parámetro que representa la capacidad del primer compactor k_p, se determinan los valores de ε asociados a esta estructura para una comparación más justa con los sketches implementados. Esto fue realizado mediante la función de la biblioteca *get normalized rank error* tras ingresar todos los datos correspondientes a los datasets de trazas. La tabla 4.1.3 indica la relación entre los valores de k_p y ε obtenidos.

4.2. Espacio y tiempo utilizado por los sketches

4.2.1. Descripción del experimento

Este experimento consiste en medir el espacio y tiempo ocupado por los sketches implementados. El objetivo es comparar estas medidas entre sketches con parámetros equivalentes, y para medir cómo afecta el valor de los parámetros en el espacio y tiempo de construcción empleado por las estructuras. Específicamente, las medidas son obtenidas tras inicializar e ingresar todos los datos del dataset respectivo. Cada prueba es repetida 15 veces para cada tipo de configuración de sketch, donde una configuración esta compuesta por una combinación de los parámetros:

- Cota de error ε: Para los sketches MRL y KLL los valores 0.002, 0.004, 0.006, 0.008 y 0.010
- Probabilidad de fallo δ (KLL): Valores entre 0.01 y 0.05
- Factor de reducción c (KLL): Valores entre 0.50 y 0.66

Los resultados obtenidos son mostrados por medio de tablas que indican el valor de los parámetros con los que se inicializan los sketches, el espacio ocupado y tanto el tiempo en segundos como su variación estándar para cada configuración.

Por otra parte, se incluye una comparación del tiempo ocupado en realizar distintas cantidades de consultas tipo Rank individual, Rank grupal y Select mediante el uso de sketches creados con parámetros ε, δ y c correspondientes a 0.005, 0.01 y 0.66 respectivamente. Adicionalmente, se compara el rendimiento del uso de búsqueda lineal y binaria en el arreglo temporal generado durante la consulta Select.

4.2.2. Resultados

Los resultados extraídos por los sketches MRL y KLL en los datasets de Chicago-20150219, Mendeley y Mawi-20191102 se encuentran en las tablas 4.2.1, 4.2.2 y 4.2.3 respectivamente. De estos se puede observar que, por lo general, KLL emplea un espacio de aproximadamente un orden de magnitud menor que MRL para las configuraciones usadas. Además, muestra que el espacio del sketch KLL es más sensible a cambios en el valor de ε que en δ, lo que ocurre para cada dataset utilizado. Es importante mencionar que el espacio empleado por los sketches con tuplas solo debe añadir el tamaño asignado para el min-heap, de manera que la diferencia de tamaño entre el KLLTuple y MRLTuple es la misma. Por otra parte, el tiempo empleado por el sketch MRL es mayor al KLL. Esto ocurre porque el MRL no realiza un muestreo de los datos, por lo que necesita un mayor número de compactaciones asociados a los compactores de menor nivel. Cabe destacar que la variación de tiempo de ejecución para cada configuración es baja, indicando que tanto el espacio como tiempo ocupado por cada sketch es consistente entre sí.

ε	δ	c	KLL[B]	MRL[B]	T_{KLL}[s]	$\sigma_{TKLL[s]}$	T_{MRL}[s]	$\sigma_{TMRL[s]}$
0.002	0.01	0.50	65948	720340	9.40	0.0285	18.12	0.1031
0.002	0.01	0.66	75852	720340	9.42	0.2005	18.12	0.1031
0.002	0.05	0.50	51076	720340	9.38	0.0045	18.12	0.1031
0.002	0.05	0.66	58636	720340	9.35	0.1460	18.12	0.1031
0.004	0.01	0.50	33156	416356	7.49	0.0061	17.51	0.0766
0.004	0.01	0.66	37812	416356	7.67	0.2212	17.51	0.0766
0.004	0.05	0.50	25732	416356	7.48	0.0063	17.51	0.0766
0.004	0.05	0.66	29244	416356	7.50	0.0725	17.51	0.0766
0.006	0.01	0.50	18668	317668	7.17	0.0077	17.50	0.0172
0.006	0.01	0.66	21612	317668	7.26	0.1448	17.50	0.0172
0.006	0.05	0.50	14556	317668	7.16	0.0027	17.50	0.0172
0.006	0.05	0.66	19532	317668	7.24	0.0954	17.50	0.0172
0.008	0.01	0.50	16740	238372	7.01	0.0147	17.19	0.0391
0.008	0.01	0.66	16092	238372	7.03	0.0198	17.19	0.0391
0.008	0.05	0.50	10948	238372	7.01	0.0117	17.19	0.0391
0.008	0.05	0.66	14564	238372	7.03	0.0155	17.19	0.0391
0.010	0.01	0.50	11348	201972	6.92	0.0041	16.78	0.0454
0.010	0.01	0.66	12908	201972	7.05	0.1926	16.78	0.0454
0.010	0.05	0.50	10620	201972	6.92	0.0032	16.78	0.0454
0.010	0.05	0.66	10044	201972	6.98	0.0413	16.78	0.0454

Tabla 4.2.1: Espacio en bytes y tiempo en segundos empleado por los sketches con distintos parámetros en Chicago-20150219. Donde ε corresponde a la cota de error tolerada, δ a la probabilidad fallo, c corresponde al factor de reducción empleado y σ_T a la desviación estándar del tiempo ocupado.

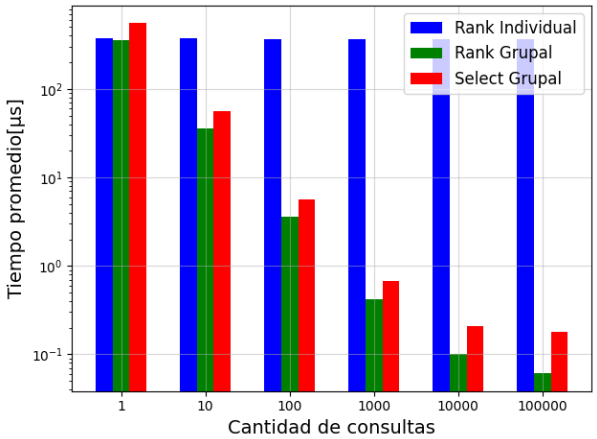
ε	δ	c	KLL[B]	MRL[B]	T_{KLL}[s]	$\sigma_{TKLL[s]}$	T_{MRL}[s]	$\sigma_{TMRL[s]}$
0.002	0.01	0.50	55012	520308	2.68	0.0009	2.90	0.0258
0.002	0.01	0.66	64916	520308	2.58	0.0521	2.90	0.0258
0.002	0.05	0.50	42628	520308	2.67	0.0064	2.90	0.0258
0.002	0.05	0.66	58636	520308	2.60	0.0395	2.90	0.0258
0.004	0.01	0.50	27684	280308	1.54	0.0033	2.84	0.0297
0.004	0.01	0.66	32340	280308	1.53	0.0212	2.84	0.0297
0.004	0.05	0.50	21508	280308	1.53	0.0008	2.84	0.0297
0.004	0.05	0.66	25020	280308	1.52	0.0225	2.84	0.0297
0.006	0.01	0.50	22316	224308	1.53	0.0038	2.80	0.0149
0.006	0.01	0.66	21612	224308	1.56	0.0389	2.80	0.0149
0.006	0.05	0.50	17372	224308	1.54	0.0010	2.80	0.0149
0.006	0.05	0.66	19532	224308	1.55	0.0319	2.80	0.0149
0.008	0.01	0.50	14004	180340	1.24	0.0046	2.77	0.0321
0.008	0.01	0.66	16092	180340	1.24	0.0055	2.77	0.0321
0.008	0.05	0.50	10948	180340	1.24	0.0012	2.77	0.0321
0.008	0.05	0.66	12452	180340	1.24	0.0058	2.77	0.0321
0.010	0.01	0.50	13540	144340	1.24	0.0046	2.73	0.0335
0.010	0.01	0.66	12908	144340	1.24	0.0068	2.73	0.0335
0.010	0.05	0.50	10620	144340	1.24	0.0042	2.73	0.0335
0.010	0.05	0.66	11740	144340	1.25	0.0086	2.73	0.0335

Tabla 4.2.2: Espacio en bytes y tiempo en segundos empleado por los sketches con distintos parámetros en Mendeley. Donde ε corresponde a la cota de error tolerada, δ a la probabilidad fallo, c corresponde al factor de reducción empleado y σ_T a la desviación estándar del tiempo ocupado.

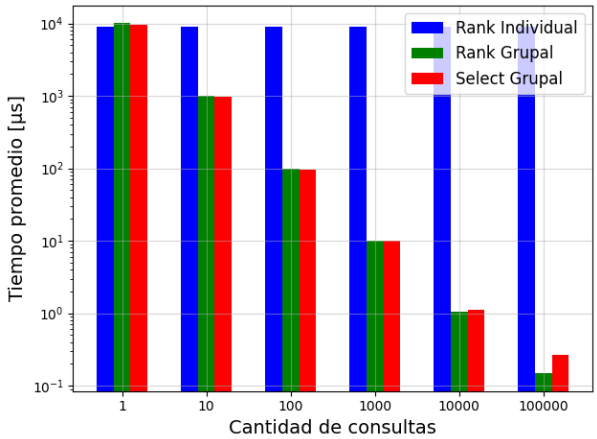
ε	δ	c	KLL[B]	MRL[B]	T_{KLL}[s]	$\sigma_{TKLL[s]}$	T_{MRL}[s]	$\sigma_{TMRL[s]}$
0.002	0.01	0.50	65948	952372	29.27	0.0195	69.10	0.2119
0.002	0.01	0.66	75852	952372	29.03	0.0308	69.10	0.2119
0.002	0.05	0.50	51076	952372	29.27	0.0261	69.10	0.2119
0.002	0.05	0.66	58636	952372	29.07	0.0362	69.10	0.2119
0.004	0.01	0.50	33156	540388	27.40	0.0097	67.04	0.1428
0.004	0.01	0.66	37812	540388	27.41	0.0458	67.04	0.1428
0.004	0.05	0.50	25732	540388	27.40	0.0252	67.04	0.1428
0.004	0.05	0.66	29244	540388	27.44	0.0732	67.04	0.1428
0.006	0.01	0.50	22316	405652	27.09	0.0124	66.01	0.2631
0.006	0.01	0.66	21612	405652	27.12	0.0309	66.01	0.2631
0.006	0.05	0.50	17372	405652	27.08	0.0100	66.01	0.2631
0.006	0.05	0.66	19532	405652	27.14	0.0276	66.01	0.2631
0.008	0.01	0.50	16740	304404	26.93	0.0224	65.91	0.0210
0.008	0.01	0.66	18828	304404	27.01	0.1356	65.91	0.0210
0.008	0.05	0.50	13060	304404	26.92	0.0093	65.91	0.0210
0.008	0.05	0.66	14564	304404	26.99	0.0437	65.91	0.0210
0.010	0.01	0.50	11348	240388	26.85	0.0116	64.54	0.1660
0.010	0.01	0.66	12908	240388	26.92	0.0688	64.54	0.1660
0.010	0.05	0.50	08924	240388	26.85	0.0138	64.54	0.1660
0.010	0.05	0.66	10044	240388	26.90	0.0332	64.54	0.1660

Tabla 4.2.3: Espacio en bytes y tiempo en segundos empleado por los sketches con distintos parámetros en Mawi-20191102. Donde ε corresponde a la cota de error tolerada, δ a la probabilidad fallo, c corresponde al factor de reducción empleado y σ_T a la desviación estándar del tiempo ocupado.

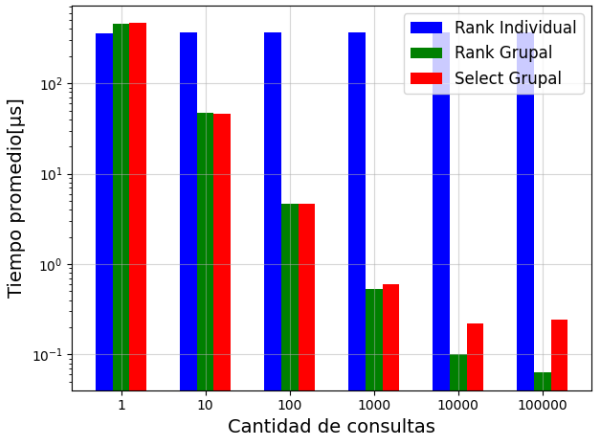
Los tiempos promedios de ejecución de las operaciones Rank individual, Rank grupal y Select se encuentran en la Figura 4.2.1, donde se observa que para Rank individual el tiempo promedio se mantiene constante independiente del número de consultas realizadas, lo que ocurre debido a que cada elemento realiza el mismo procedimiento. Por otro lado, el tiempo promedio de las consultas Rank grupal y Select va disminuyendo acorde aumenta el número de consultas. Este comportamiento es esperado, ya que estas operaciones generan un arreglo temporal que permite responder de manera rápida cada consulta. Respecto a este arreglo, se verifica el tiempo que tarda realizar búsqueda binaria y lineal para un distinto número de consultas de la operación Select, obteniendo los resultados encontrados en la Figura 4.2.2, en el que se encuentra que búsqueda binaria es 100 veces menos en cada caso. Es importante destacar que la experimentación realizada no considera el tiempo ocupado en la construcción del arreglo de la consulta, por lo que se recomienda el uso de búsqueda binaria a excepción de cuando se quiere realizar una cantidad baja de consultas con pocos datos en el sketch.



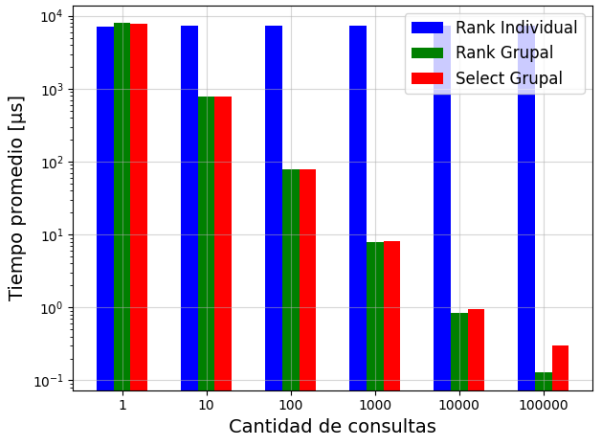
(a) Dataset Chicago-20150219 mediante el uso de KLL.



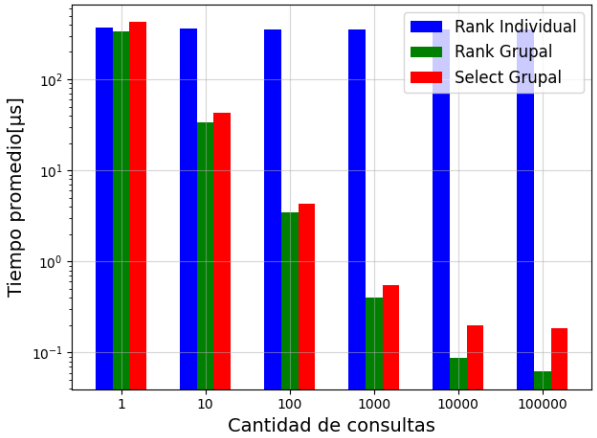
(b) Dataset Chicago-20150219 mediante el uso de MRL.



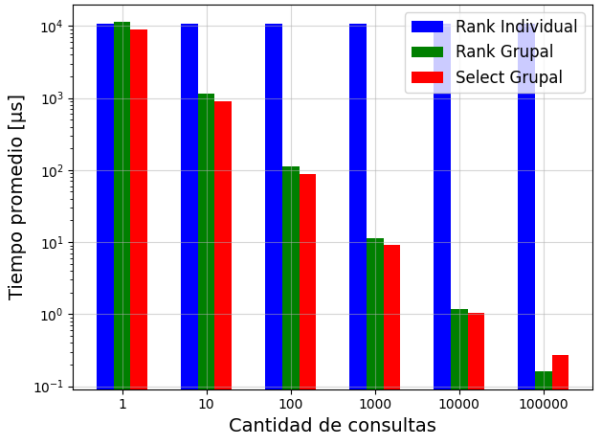
(c) Dataset Mendeley mediante el uso de KLL.



(d) Dataset Mendeley mediante el uso de MRL.

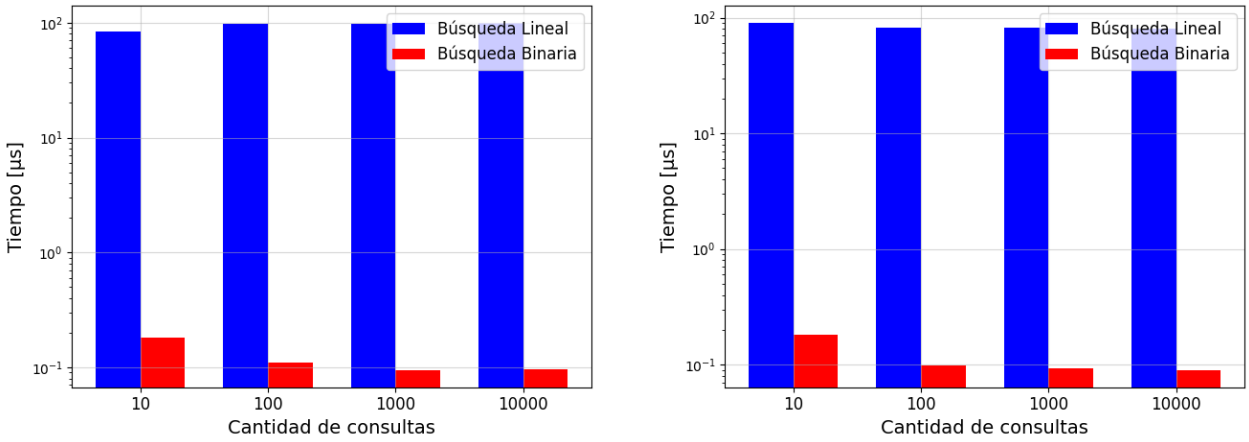


(e) Dataset Mawi-20191102 mediante el uso de KLL.



(f) Dataset Mawi-20191102 mediante el uso de MRL.

Figura 4.2.1: Gráficos de tiempo promedio ocupado por consultas a las operaciones Rank individual, Rank grupal y Select empleando un sketch KLL con $\varepsilon = 0.005$, $\delta = 0.01$ y $c = 0.66$ y un sketch MRL con $\varepsilon = 0.005$ en los datasets Chicago-20150219, Mendeley y Mawi-20191102.



(a) Dataset Chicago-20150219.

(b) Dataset Mawi-20191102.

Figura 4.2.2: Gráfico de tiempo promedio ocupado en realizar búsqueda lineal o binaria en el arreglo generado por consultas Select en un sketch KLL con $\varepsilon = 0.005$ y $\delta = 0.01$ en los datasets Chicago-20150219 y Mawi-20191102.

4.3. Estimación del rank de los sketches

4.3.1. Descripción del experimento

Este experimento busca estudiar el rendimiento de los sketches respecto a la consulta *Rank*, observando si las cotas proporcionadas al momento de inicializar las estructuras son respetadas. Asimismo, se quiere conocer el impacto de los parámetros utilizados en relación con las estimaciones obtenidas.

El experimento consta de la extracción del *Rank* real y estimado asociado a los elementos ubicados en los percentiles del dataset, es decir, aquellos elementos que dividen el 1 %, 2 %, ..., 99 % de los datos. Los sketches empleados son MRL y KLL donde cada experimento se repite 15 veces para cada una de las configuraciones utilizadas. Dichas configuraciones se encuentran determinadas por los siguientes parámetros:

- Cota de error ε: Para los sketches MRL y KLL los valores 0.002, 0.003, 0.004, 0.005, 0.006, 0.007, 0.008, 0.009, 0.010
- Probabilidad de fallo δ (KLL): 0.01
- Factor de reducción c (KLL): 0.50 y 0.66

El error absoluto es una métrica utilizada para comparar el rendimiento de las consultas tipo *Rank*, la cual indica qué tan cercano se encuentra una estimación del verdadero valor. Matemáticamente, el error absoluto se define como la diferencia entre el valor real de una medida X y el valor de su estimación $\hat{X}$, en el contexto de la consulta realizada corresponde a la diferencia entre la posición real de un elemento y su posición estimada. Para facilitar comparaciones entre datasets de diferentes tamaños, se opta por normalizar el error absoluto,

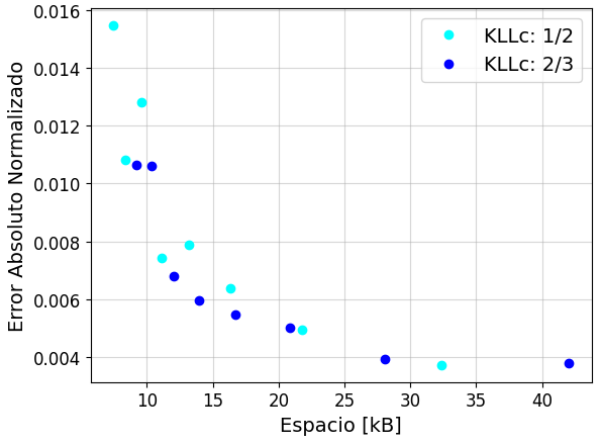
dividiendo el valor obtenido por la cantidad de elementos encontrados en el dataset, lo que produce un valor en el intervalo [0, 1]. La fórmula para el error absoluto normalizado E_{an} se expresa en la ecuación 4.2.

$$E_{an} = \frac{|X - \hat{X}|}{n} \tag{4.2}$$

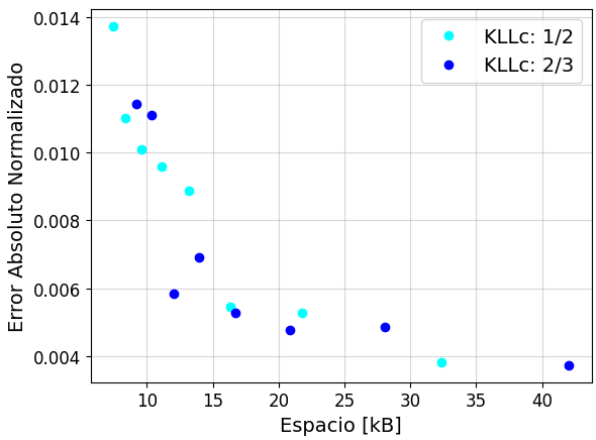
Los resultados consideran un promedio del error absoluto normalizado respecto a aquellos elementos ubicados en los percentiles de cada dataset. Se presentan tres tipos de gráficos. El primero compara el error absoluto normalizado en función del espacio utilizado por dos sketches KLL que emplean un distinto valor del parámetro c. El segundo compara el rendimiento para las distintas configuraciones de los sketches MRL, KLL con $c = \frac{2}{3}$ y ApacheDatasketch. Es relevante indicar que para estos dos tipos de gráficos las configuraciones que emplean un mayor valor de ε siempre utilizarán una mayor cantidad de espacio. El último tipo de gráfico representa el error absoluto normalizado en función del tiempo de construcción de cada sketch.

4.3.2. Resultados

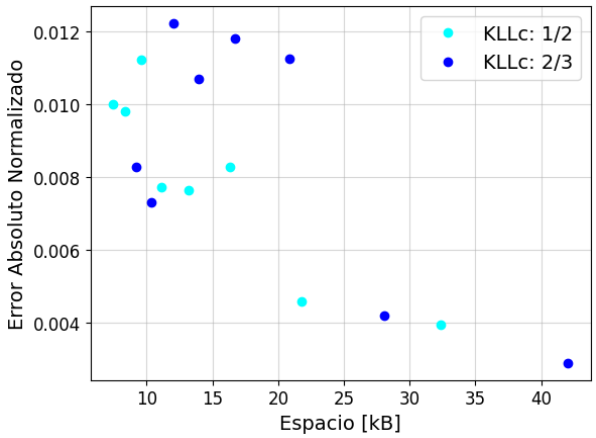
La Figura 4.3.1 muestra la diferencia del error absoluto normalizado para sketches KLL que se distinguen a partir de su factor de reducción, donde principalmente se obtienen mejores resultados tras usar $c = \frac{2}{3}$. También se observa que un mayor uso del espacio por lo general involucra un menor error absoluto normalizado. No obstante, estas observaciones no son ciertas únicamente en el dataset de Mendeley. Puesto que el espacio usado por cada sketch depende del valor de ε definido, se puede contemplar que la selección de una cota más estricta mejora el rendimiento de los sketches.



(a) Dataset Chicago-20150219



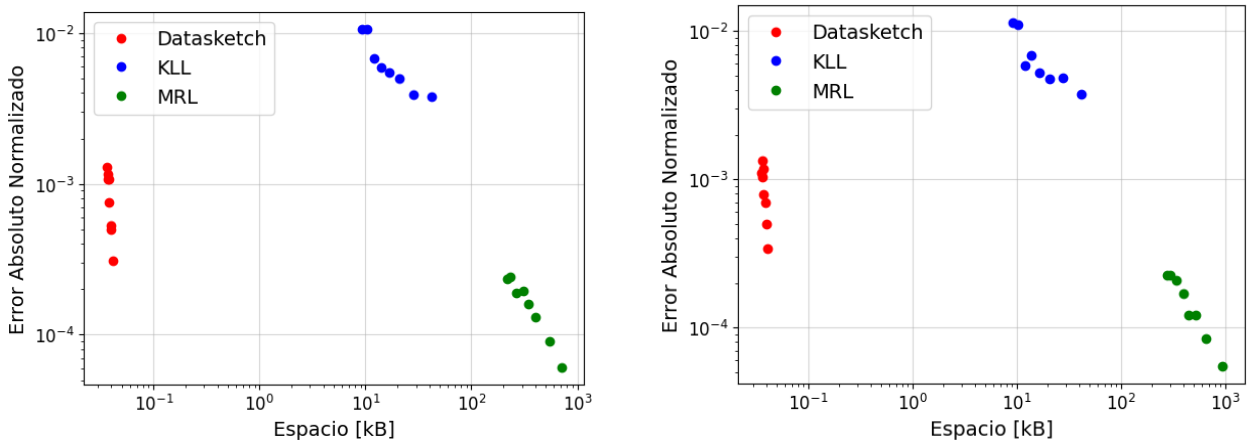
(b) Dataset Mawi-20191102



(c) Dataset Mendeley

Figura 4.3.1: Gráficos de error absoluto normalizado respecto al espacio ocupado por distintas configuraciones del sketch KLL con parámetros $c = \frac{1}{2}$ y $\frac{2}{3}$.

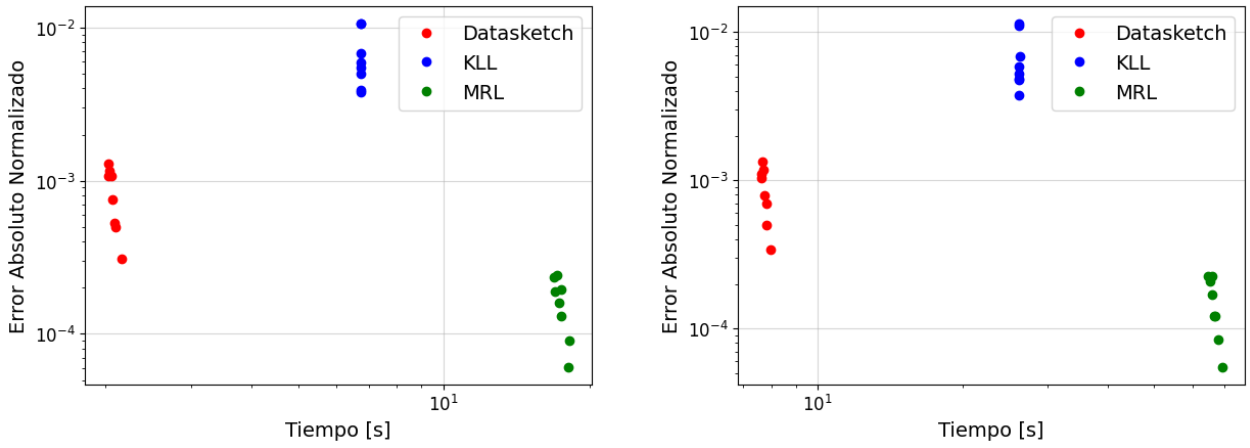
La Figura 4.3.2 contiene gráficos que comparan el rendimiento de los distintos sketches, permite apreciar que las configuraciones del KLL emplean aproximadamente el mismo tamaño que ApacheDatasketch, aunque estos últimos obtienen una diferencia cercana a un orden de magnitud menor respecto a la métrica de error absoluto normalizado en todos los casos. A diferencia de los otros sketches, el MRL obtiene valores más altos de la métrica medida al coste de emplear una cantidad de espacio significativamente mayor que las otras dos estructuras.



(a) Dataset Chicago-20150219 (b) Dataset Mawi-20191102

Figura 4.3.2: Gráficos de error absoluto normalizado respecto al espacio ocupado por distintas configuraciones de los sketches KLL, MRL y ApacheDatasketch.

Por su parte, la Figura 4.3.3 demuestra que el tiempo de ejecución por cada sketch no es un buen indicador para determinar el valor del error absoluto normalizado, puesto que existe poca variación en el tiempo empleado por cada sketch. La diferencia entre los valores de tiempo de ejecución obtenidos radica principalmente en la inclusión de una tasa de muestreo y la implementación propia de cada sketch.



(a) Dataset Chicago-20150219 (b) Dataset Mawi-20191102

Figura 4.3.3: Gráficos de error absoluto normalizado respecto al tiempo ocupado por distintas configuraciones de los sketches KLL, MRL y ApacheDatasketch.

4.4. Estimación de elementos en los sketches MRL y KLL

4.4.1. Descripción del experimento

Una de las consultas alternativas que responden los sketches consiste en estimar un elemento mediante la operación *Select* o *Cuantil*. Este experimento busca determinar qué tan cercanos están esos elementos a los percentiles reales en los datasets utilizados. Las pruebas fueron realizadas con dos tipos de datasets, sintéticos y reales. Los datasets sintéticos son generados a partir de dos distribuciones, una de ellas corresponde a una distribución uniforme con elementos dentro del rango $[1; 100{,}000]$. La otra distribución, la cual se nombra como "Ggap", tiene un único gran salto entre el valor de un elemento y otro, esto fue representado mediante dos distribuciones gaussianas. Una de ellas almacena el 70 % de los datos con media 20,000 y desviación estándar 5,000, mientras que el 30 % restante tienen 80,000 y 10,000 respectivamente. Los datasets con trazas reales para este experimento consisten en Chicago-20150219 y Mawi-20191102.

La ejecución del experimento es similar a la estimación de los ranks, donde cada tipo de sketch ingresa los datos provenientes de los datasets respectivos y responde a la consulta de *Cuantil* para cada uno de los percentiles. Cada experimento es repetido 15 veces con las siguientes configuraciones para los sketches:

- Cota de error ε: Valores 0.003 y 0.007
- Probabilidad de fallo δ (KLL): 0.01
- Factor de reducción c (KLL): 0.66

El rendimiento de las estimaciones se evalúa por el error relativo E_r, métrica que indica qué tan cercano o lejano se encuentra el valor estimado en relación al valor real. El error relativo se calcula como la proporción entre la diferencia del valor real de una medida X con el valor de su estimación $\hat{X}$, respecto al valor real de dicha medida. La fórmula para su obtención se encuentra en 4.3. Se opta utilizar esta métrica sobre error absoluto debido a que esta última no considera la magnitud del valor real, por lo que una gran diferencia en el valor de la estimación puede tener un pequeño impacto en comparación al valor del elemento o viceversa.

$$E_r = \frac{|X - \hat{X}|}{X} \tag{4.3}$$

Los resultados son indicados en dos tipos de gráficos. El primero de ellos representa al error relativo asociado a las estimaciones, observando el detalle para distintas vistas como los primeros y últimos 10 percentiles, junto a una vista general de los percentiles para cada tipo de sketch. El segundo compara los valores de los elementos encontrados en los percentiles de los datasets y sus estimaciones.

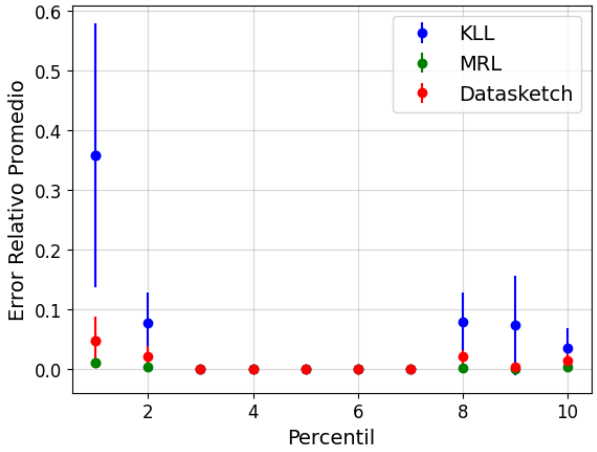
4.4.2. Resultados

Los gráficos 4.4.1 y 4.4.2 muestran el valor del error relativo promedio asociado a las estimaciones de cada elemento en las distintas iteraciones efectuadas en los datasets reales. Los resultados muestran que el error relativo promedio de cada sketch es mayor en los primeros percentiles de cada dataset, y va disminuyendo a medida que aumenta el valor del percentil consultado. Por otra parte, los resultados indican que el sketch KLL estima de peor manera los elementos en comparación a los otros sketches. Asimismo, el sketch MRL contiene las mejores estimaciones para cada caso. Por otro lado, ApacheDatasketch tiene valores de la métrica mucho menor que KLL usando espacios similares. Esto se repite en todos los datasets usados. Es relevante destacar que en todos estos casos las modificaciones en la cota ε no afectan de manera significativa los resultados.

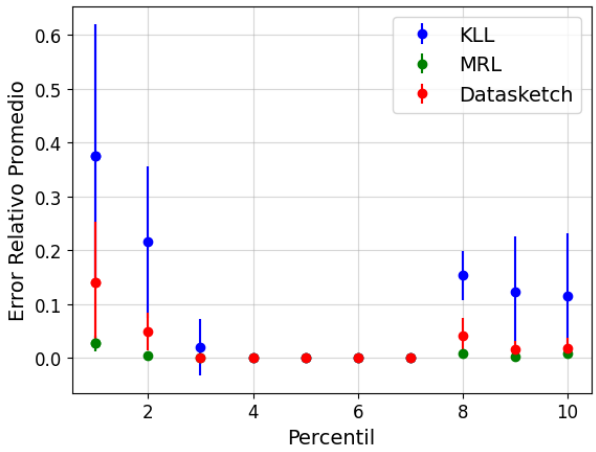
La vista de los primeros y últimos percentiles indica que a las tres estructuras empleadas se les dificulta detectar correctamente los elementos encontrados en los bordes de los datasets. Aún así, el KLL tiene un mayor error promedio que el resto durante la estimación. Esto puede deberse a que este es el único sketch entre los tres que incorpora una tasa de muestreo, de tal forma que no asegura mantener almacenado los valores ubicados a los extremos de la distribución en cada iteración.

Las Figuras 4.4.3 y 4.4.4 indican los resultados obtenidos para los datasets sintéticos, donde se destaca que para la distribución Ggap existe una dificultad en la estimación de los elementos cercanos al percentil donde ocurre un gran salto entre el valor de un elemento y otro. Esto no ocurre para el caso de la distribución uniforme.

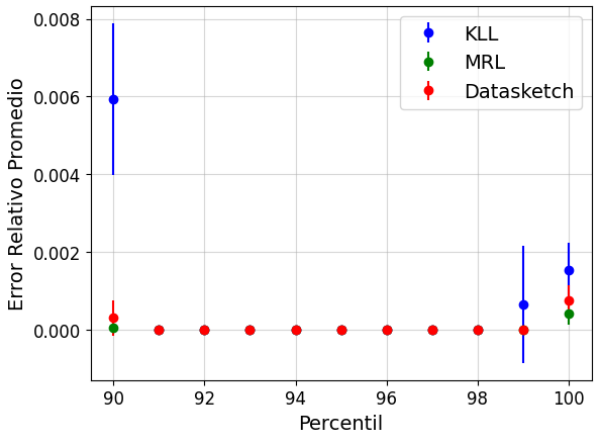
Es notorio mencionar que se dificulta emplear el uso del error relativo para comparar el rendimiento de los sketches, esto es debido a que existe una gran diferencia entre los valores de los elementos encontrados en los primeros y últimos percentiles, por lo que una misma diferencia absoluta entre un valor y otro altera de mayor forma el valor de la métrica en un cuantil determinado en comparación a otro. Un ejemplo de ello se puede evidenciar comparando el promedio del valor de la diferencia de los elementos encontrados en el salto existente de la distribución Ggap con la estimación del último percentil y el error relativo obtenido en las Figuras 4.4.5 y 4.4.4



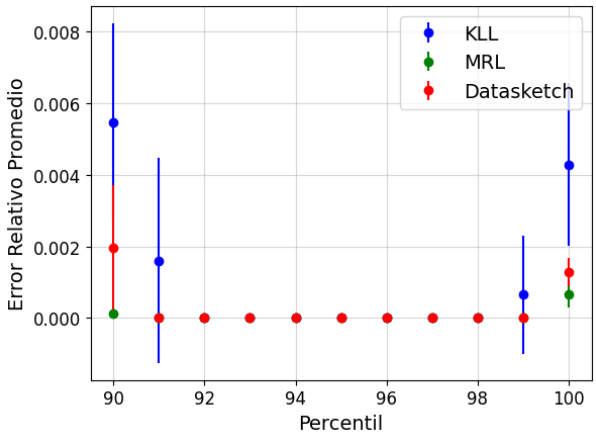
(a) Primeros 10 percentiles con $\varepsilon = 0.003$



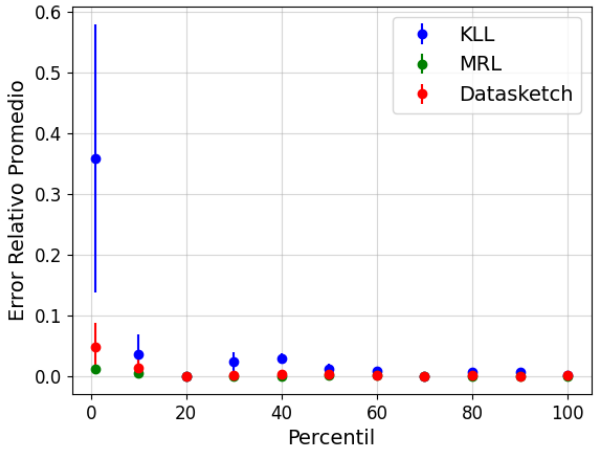
(b) Primeros 10 percentiles con $\varepsilon = 0.007$



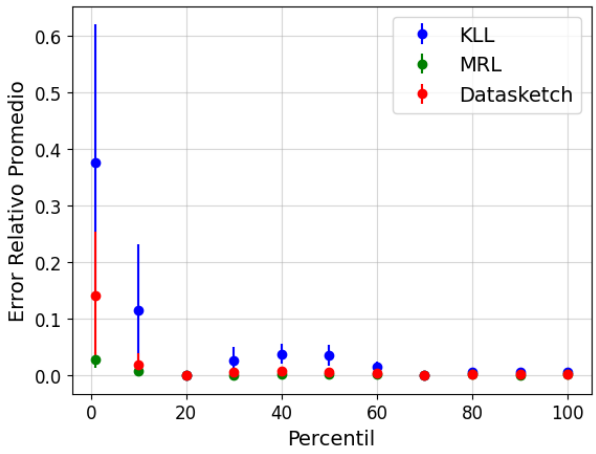
(c) Últimos 10 percentiles con $\varepsilon = 0.003$



(d) Últimos 10 percentiles con $\varepsilon = 0.007$

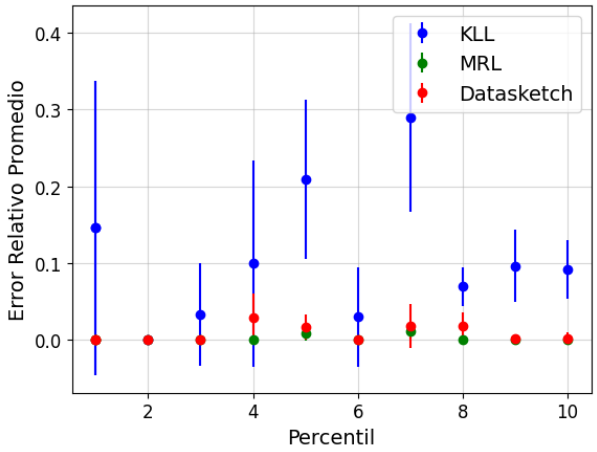


(e) Deciles con $\varepsilon = 0.003$

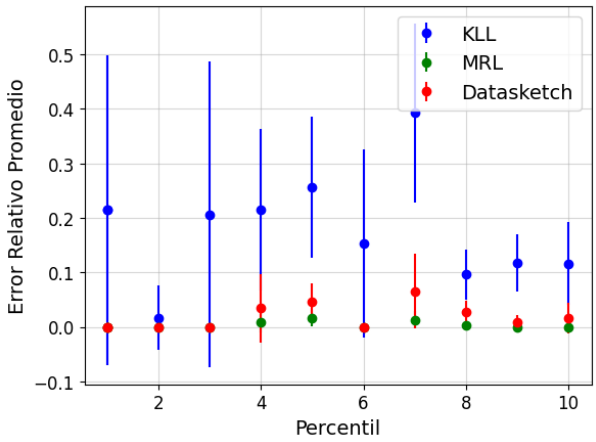


(f) Deciles con $\varepsilon = 0.007$

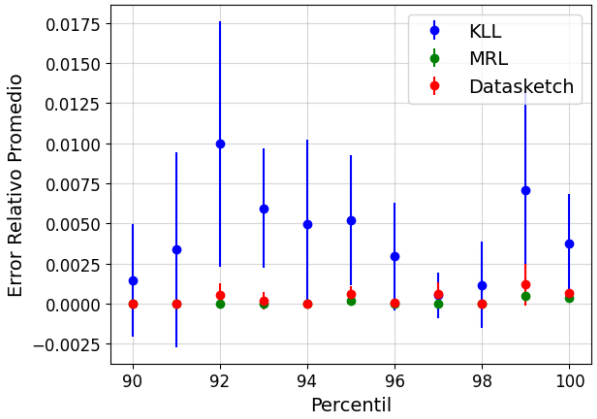
Figura 4.4.1: Gráficos que indican el error relativo promedio de 15 iteraciones para la estimación de elementos ubicados en los percentiles del dataset Chicago-20150219, donde se ocupan sketches KLL con $\delta = 0.01$; además, estos utilizan un espacio aproximado de 28 y 12 KB, mientras que los sketches MRL ocupan 541 y 266 KB para valores de ε de 0.003 y 0.007 respectivamente.



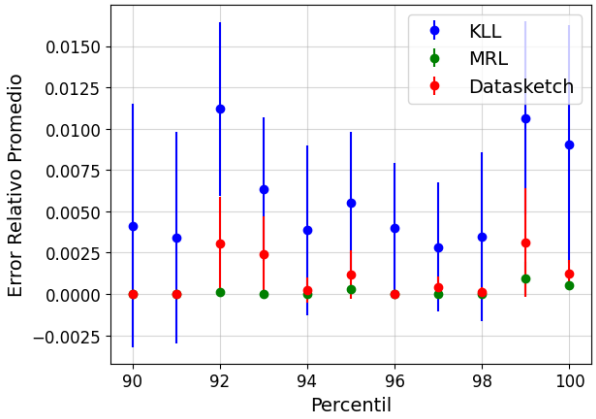
(a) Primeros 10 percentiles con $\varepsilon = 0.003$



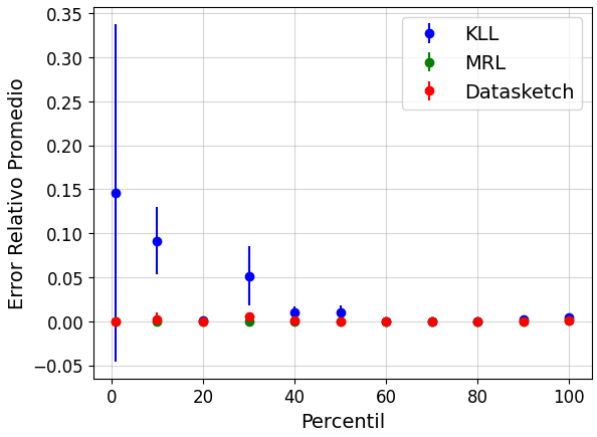
(b) Primeros 10 percentiles con $\varepsilon = 0.007$



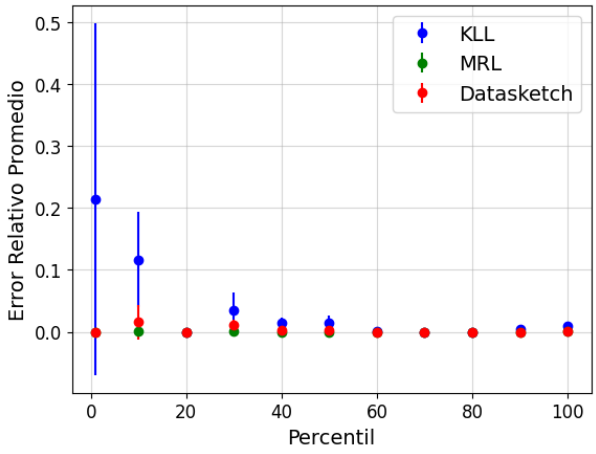
(c) Últimos 10 percentiles con $\varepsilon = 0.003$



(d) Últimos 10 percentiles con $\varepsilon = 0.007$

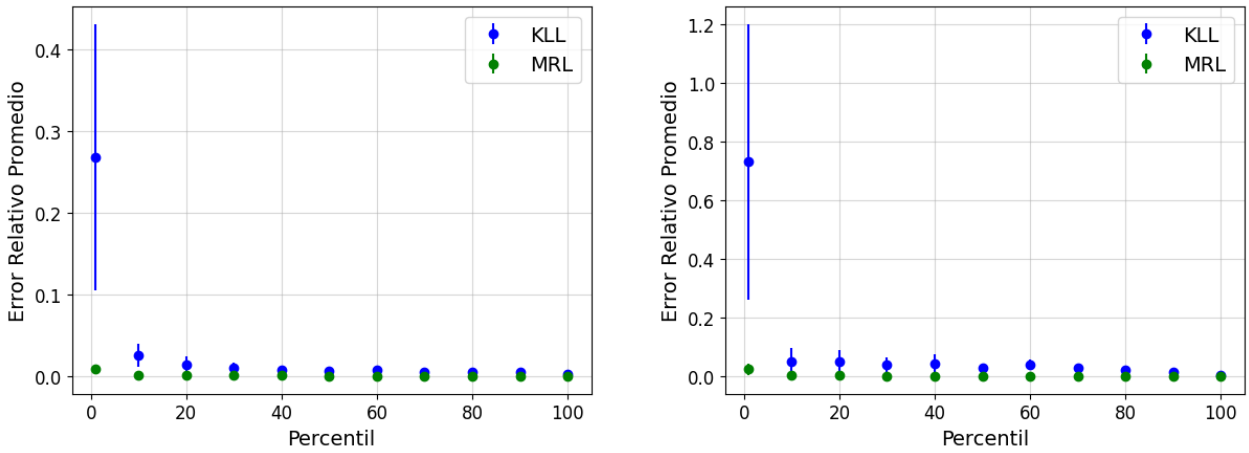


(e) Deciles con $\varepsilon = 0.003$



(f) Deciles con $\varepsilon = 0.007$

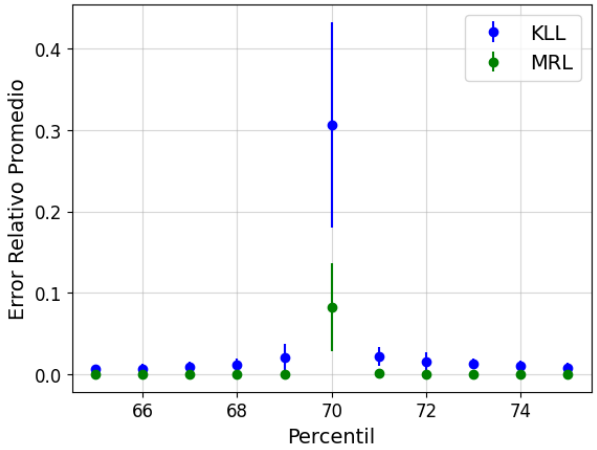
Figura 4.4.2: Gráficos que indican el error relativo promedio de 15 iteraciones para la estimación de elementos ubicados en los percentiles del dataset Mawi-20191102, donde se ocupan sketches KLL con $\delta = 0.01$; además, estos utilizan un espacio aproximado de 28 y 12 KB, mientras que los sketches MRL ocupan 656 y 339 KB para valores de ε de 0.003 y 0.007 respectivamente.



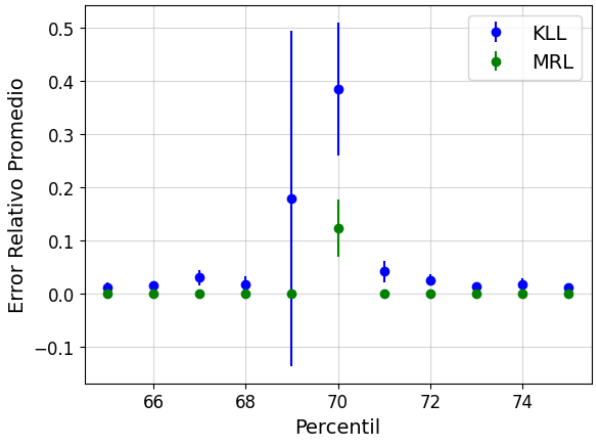
(a) Deciles con $\varepsilon = 0.003$

(b) Deciles con $\varepsilon = 0.007$

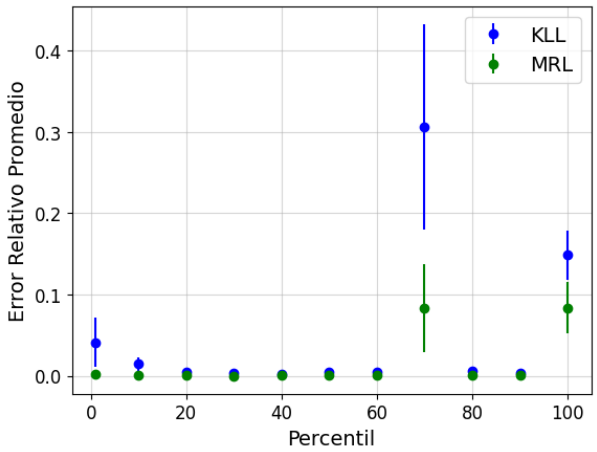
Figura 4.4.3: Gráficos que indican el error relativo promedio de 15 iteraciones para la estimación de elementos ubicados en los percentiles del dataset sintético uniforme con 10000000 elementos, donde se ocupan sketches KLL con $\delta = 0.01$; además, estos utilizan un espacio aproximado de 28 y 12 KB, mientras que los sketches MRL ocupan 469 y 246 KB para valores de ε de 0.003 y 0.007 respectivamente.



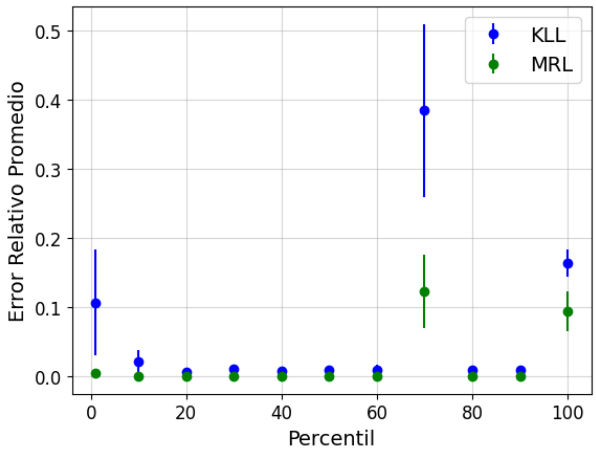
(a) Percentiles 65-75 con $\varepsilon = 0.003$



(b) Percentiles 65-75 con $\varepsilon = 0.007$

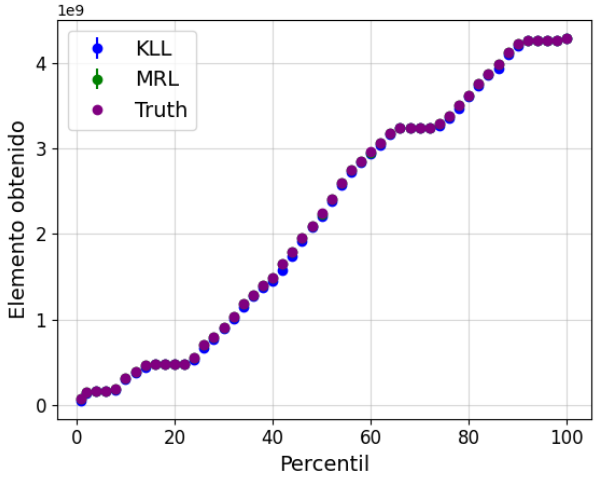


(c) Deciles con $\varepsilon = 0.003$

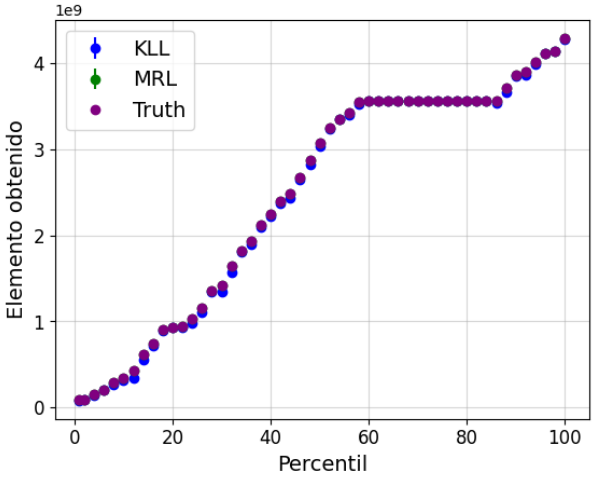


(d) Deciles con $\varepsilon = 0.007$

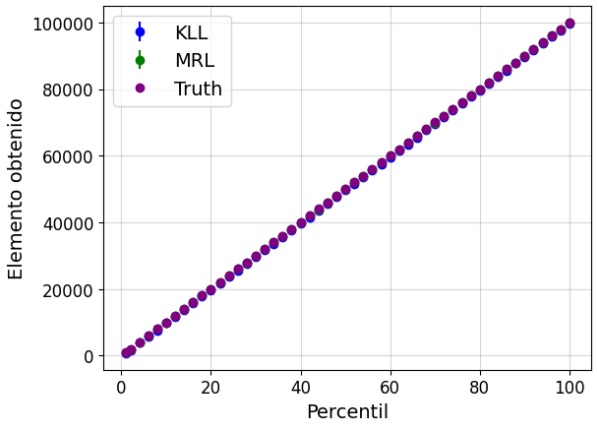
Figura 4.4.4: Gráficos que indican el error relativo promedio de 15 iteraciones para la estimación de elementos ubicados en los percentiles del dataset sintético Ggap con 10000000 elementos, donde se ocupan sketches KLL con $\delta = 0.01$; además, estos utilizan un espacio aproximado de 28 y 12 KB, mientras que los sketches MRL ocupan 469 y 246 KB para valores de ε de 0.003 y 0.007 respectivamente.



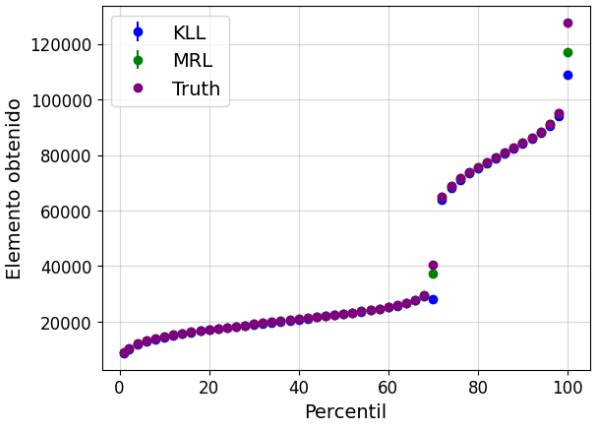
(a) Elementos estimados en Chicago-20150219



(b) Elementos estimados en Mawi-20191102



(c) Elementos estimados en dataset uniforme



(d) Elementos estimados en dataset Ggap

Figura 4.4.5: Estimación de elementos encontrados en los percentiles de cada dataset mediante sketches KLL y MRL con $\varepsilon = 0.003$ y $\delta = 0.01$; Los datasets utilizados corresponden a Chicago-20150219, Mawi-20191102, uniforme y Ggap con 10000000 de elementos cada uno, donde todos los sketches KLL ocupan un espacio aproximado de 28 KB, mientras que los MRL utilizan 542, 656, 469 y 469 KB respectivamente.

4.5. Estimación de los Top-K flujos con mayor payload

4.5.1. Descripción del experimento

Una de las funcionalidades incorporadas por los sketches con tuplas consiste en estimar aquellos flujos con mayor payload en una red y de indicar cuál es el payload asociado para cada uno de los flujos rescatados. Este experimento quiere comprobar qué tan efectivo se pueden estimar los top-k elementos de un dataset mediante la operación *TopFlows*, recordando que el valor de K está dado por el tamaño del heap usado. El experimento consiste en ingresar todos los elementos del dataset en el sketch para posteriormente extraer aquellos flujos con mayor payload y sus valores asociados por medio de la consulta *TopFlows*. Cada experimento fue reiterado 5 veces para cada configuración del sketch KLL, determinadas por los parámetros:

- ε: 0.003, 0.005, 0.007 y 0.009
- δ: 0.001
- c: 0.66
- $heapElements$: 15, 63 y 255

Para detectar si los flujos estimados representan a aquellos con mayor payload, se emplea la métrica de precisión y recall. Estas métricas hacen diferencia entre los flujos, los cuales se encuentran categorizados como:

1. Verdaderos Positivos (TP): Representan aquellos flujos que forman parte de la estimación del sketch que pertenecen a los K flujos reales con mayor payload.
2. Falsos Positivos (FP): Representan aquellos flujos que forman parte de la estimación del sketch que no pertenecen a los K flujos reales con mayor payload.
3. Falsos Negativos (FN): Representan aquellos flujos que no fueron estimados por el sketch que pertenecen a los K flujos reales con mayor payload.

La precisión mide la exactitud de las predicciones positivas obtenidas, esto es realizado comparando el número de verdaderos positivos respecto al número total de predicciones positivas. Por otra parte, el recall indica la proporción de los datos que son estimados correctamente como positivos en comparación con el número real de instancias positivas existentes. La fórmula de cada métrica se encuentra en las fórmulas 4.4 y 4.5 respectivamente. Además, se calcula el error relativo medio de los flujos encontrados como verdaderos positivos mediante la fórmula 4.6 , que considera el valor real y estimado de los flujos rescatados.

$$\text{Precisión} = \frac{\text{TP}}{\text{TP} + \text{FP}} \tag{4.4}$$

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}} \tag{4.5}$$

$$E_{rm} = \sum_{i=1}^{K} \frac{1}{K} \frac{|\text{Payload}_{\text{iReal}} - \text{Payload}_{\text{iEstimado}}|}{\text{Payload}_{\text{iReal}}} \tag{4.6}$$

Los resultados se encuentran de las siguientes formas:

1. Tablas: Indican los parámetros utilizados para la construcción del sketch: ε y la cantidad de tuplas permitidas por el heap (H_{size}) ordenados de menor a mayor valor, con los que se determina la precision (P), error relativo medio (E_{rm}) de los flujos detectados como verdaderos positivos y tiempo promedio en segundos para procesar el dataset en dicha configuración.
2. Gráficos: Comparan el error relativo medio en relación al valor K empleado para *Top-Flows*, es decir, la cantidad de elementos que soporta el heap o H_{size}.
3. Histogramas: Indican la cantidad de flujos detectados como verdaderos positivos según su error relativo para un valor de K específico.

4.5.2. Resultados

Las Tablas 4.5.1 y 4.5.2 muestran los resultados, los cuales permiten apreciar varias diferencias entre el sketch KLL implementado y del proveniente de la biblioteca ApacheDatasketch. La primera de ellas consiste en los valores medios de precisión obtenidos, ya que los valores del sketch propuesto son superiores a los provenientes de la biblioteca para cada configuración, teniendo valores en el rango de $[0{,}4000; 0{,}6784]$ y $[0{,}0667; 0{,}4452]$ respectivamente. Se quiere destacar que en la mayoría de los casos aumentar el valor de H_{size} implica la obtención de mejores valores de precisión en los sketches, la excepción proviene de ApacheDatasketch, donde se produce el efecto contrario en el dataset de flujosMawi2016. Por otra parte, aumentar el valor de ε incrementa el valor de las precisiones solo en el sketch implementado. Una segunda diferencia consiste en el error relativo medio calculado respecto a la estimación de los payloads proveniente de flujos, en el cual ApacheDatasketch tiene valores despreciables en contraste con el KLL implementado, cuyos resultados son más variables y se encuentran en el rango $[0{,}0069; 0{,}0982]$. Otra distinción entre los resultados obtenidos radica en el tiempo utilizado en procesar cada dataset, donde el KLL implementado tarda menos en procesar todos los datos para cada configuración en ambos dataset. Además, los resultados indican que este sketch no fluctúa tanto su valor respecto a la cantidad de elementos en el heap en comparación a la biblioteca, el cual aumenta o disminuye drásticamente el tiempo empleado.

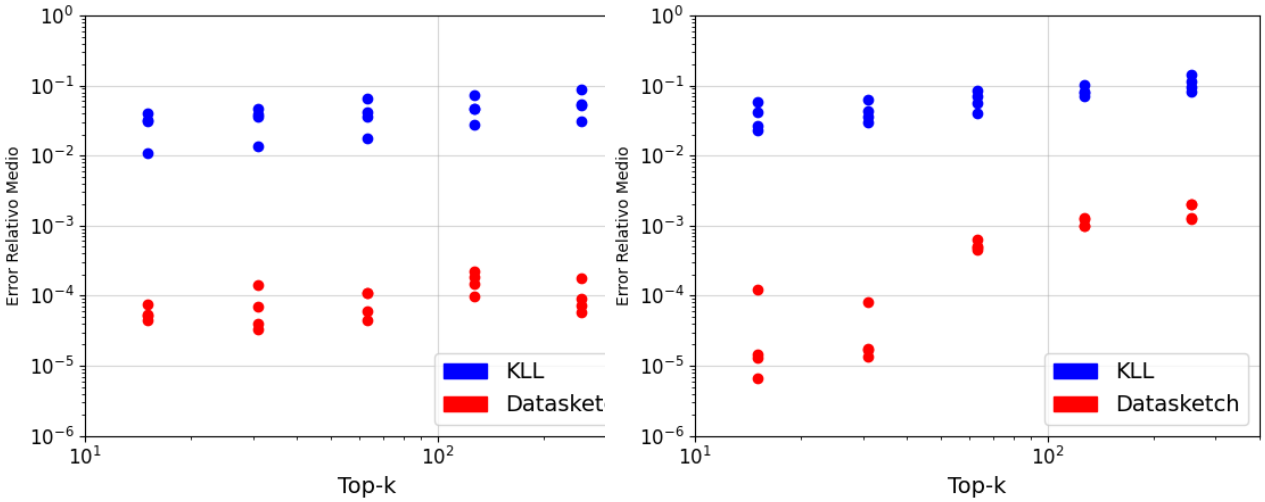
Los gráficos en la Figura 4.5.1 permiten visualizar que la implementación de ApacheDatasketch proporciona un error relativo medio que es por lo general 100 veces menor que la implementación del KLL realizado en este trabajo.

Otro resultado consiste en que ApacheDatasketch respeta la cota de ε incluida para todas las configuraciones empleadas, a diferencia del sketch implementado que en promedio no cumple con el valor de error establecido.

Los histogramas 4.5.2 y 4.5.3 permiten visualizar la cantidad de flujos detectados como verdaderos positivos para distintas configuraciones de los sketches con valores de Top-K corres-

ε	H_{size}	P_{KLL}	P_D	E_{rmKLL}	E_{rmD}	T_{KLL} (s)	T_D (s)
0.003	15	0.466	0.133	0.0119	0.0001	14.38	22.19
0.005	15	0.600	0.186	0.0072	0.0001	12.26	22.06
0.007	15	0.600	0.066	0.0069	0.0000	12.24	21.98
0.009	15	0.666	0.066	0.0234	0.0000	11.74	22.02
0.003	31	0.516	0.180	0.0103	0.0001	16.08	25.86
0.005	31	0.516	0.212	0.0095	0.0001	13.73	25.71
0.007	31	0.516	0.161	0.0075	0.0000	14.11	25.66
0.009	31	0.645	0.206	0.0231	0.0000	13.59	25.62
0.003	63	0.523	0.320	0.0081	0.0001	14.80	32.13
0.005	63	0.507	0.311	0.0142	0.0001	12.40	31.81
0.007	63	0.539	0.288	0.0139	0.0000	12.36	31.90
0.009	63	0.507	0.298	0.0323	0.0000	11.76	31.81
0.003	127	0.543	0.385	0.0102	0.0001	15.36	43.91
0.005	127	0.519	0.354	0.0200	0.0001	12.56	43.76
0.007	127	0.519	0.371	0.0199	0.0000	12.51	43.74
0.009	127	0.527	0.376	0.0461	0.0000	11.80	43.73
0.003	255	0.651	0.389	0.0161	0.0001	18.43	67.58
0.005	255	0.603	0.382	0.0305	0.0001	14.84	65.72
0.007	255	0.607	0.378	0.0303	0.0000	14.68	66.22
0.009	255	0.662	0.370	0.0639	0.0000	13.33	68.59

Tabla 4.5.1: Espacio y tiempo empleado por Datasketch y KLL con distintos parámetros en Chicago2016.



(a) Dataset flujosChicago2016.

(b) Dataset flujosMawi2016.

Figura 4.5.1: Comparación del error relativo medio según el valor de Top-K entre el sketch implementado y ApacheDatasketch para configuraciones con ε correspondientes a 0,003, 0,005, 0,007 y 0,009

ε	H_{size}	P_{KLL}	P_D	E_{rmKLL}	E_{rmD}	T_{KLL}(s)	T_D(s)
0.003	15	0.466	0.400	0.0052	0.0000	36.92	71.37
0.005	15	0.466	0.400	0.0038	0.0000	35.82	66.73
0.007	15	0.400	0.440	0.0076	0.0000	35.19	66.07
0.009	15	0.600	0.400	0.0631	0.0000	35.04	66.01
0.003	31	0.451	0.419	0.0095	0.0000	37.04	76.80
0.005	31	0.451	0.445	0.0113	0.0000	35.83	76.35
0.007	31	0.483	0.432	0.0195	0.0000	35.23	76.96
0.009	31	0.645	0.438	0.0382	0.0000	34.99	77.00
0.003	63	0.523	0.336	0.0329	0.0000	36.94	95.38
0.005	63	0.555	0.342	0.0517	0.0000	35.71	95.02
0.007	63	0.555	0.333	0.0448	0.0000	35.05	95.26
0.009	63	0.603	0.349	0.0541	0.0000	34.78	94.57
0.003	127	0.559	0.310	0.0566	0.0000	37.45	131.55
0.005	127	0.590	0.307	0.0443	0.0000	36.07	128.56
0.007	127	0.606	0.307	0.0567	0.0000	35.34	126.04
0.009	127	0.645	0.308	0.0729	0.0000	35.02	130.25
0.003	255	0.564	0.277	0.0565	0.0000	38.49	215.65
0.005	255	0.588	0.275	0.0642	0.0000	37.08	206.75
0.007	255	0.619	0.274	0.0855	0.0000	36.48	193.33
0.009	255	0.678	0.274	0.0982	0.0000	36.29	203.44

Tabla 4.5.2: Espacio y tiempo empleado por Datasketch y KLL con distintos parámetros en Mawi2016.

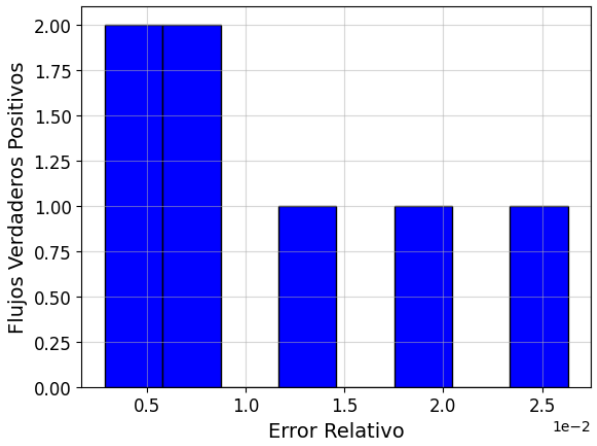
pondientes a 15, 63 y 255. De estos se vuelve a observar cómo el sketch KLLTuple detecta un mayor número de flujos verdaderos positivos que el sketch proveniente de ApacheDatasketch. Además, se destaca que para ambas estructuras la mayor cantidad de verdaderos positivos detectados tienen menores valores de error relativo dentro del sketch.

4.5.3. Análisis

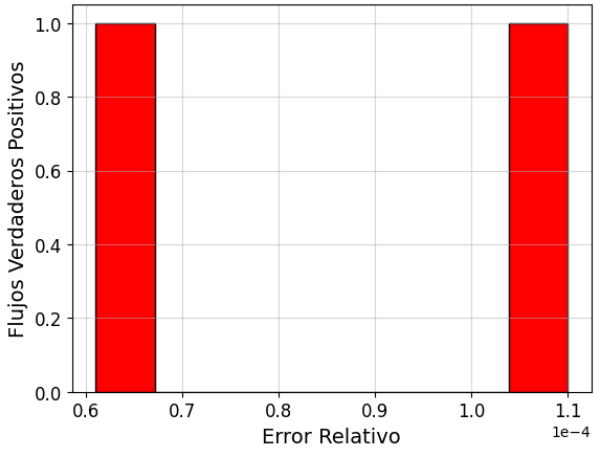
De acuerdo a los algoritmos utilizados para realizar la consulta *TopFlows*, se visualizan tres potenciales causas que podrían explicar la diferencia entre los resultados de precisión obtenidos entre los sketches utilizados:

- Comportamiento del min-heap: Dado que los sketches con tuplas solo consideran una cantidad determinada de flujos en min-heap, se pueden producir elementos que representen a máximos locales de los dataset empleados, impidiendo el paso de nuevos flujos cuyos payloads sean menores a aquellos que ya se encuentran almacenados para dicha estructura.
- Reducción de las tuplas: La reducción de tuplas permite reducir el espacio ocupado por una estructura uniendo los payloads asociados a un mismo flujo. Sin embargo, en compactores cuya capacidad es muy alta puede significar que al momento de realizar una compactación los flujos tengan payloads con gran valor, generando así máximos locales.
- Muestreo de los datos: Dependiendo de la distribución de los datos, se puede encontrar un flujo repetido varias veces para un determinado momento, lo que puede resultar en una acumulación de un gran payload para un flujo que posteriormente es enviado al min-heap.

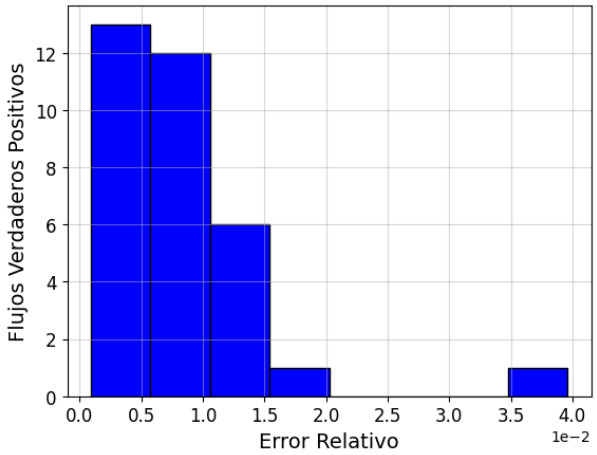
Sin embargo, estas posibles causas requieren de una verificación mediante análisis teórico y/o experimentación con una mayor cantidad de datasets que permitan validar el comportamiento de los resultados obtenidos.



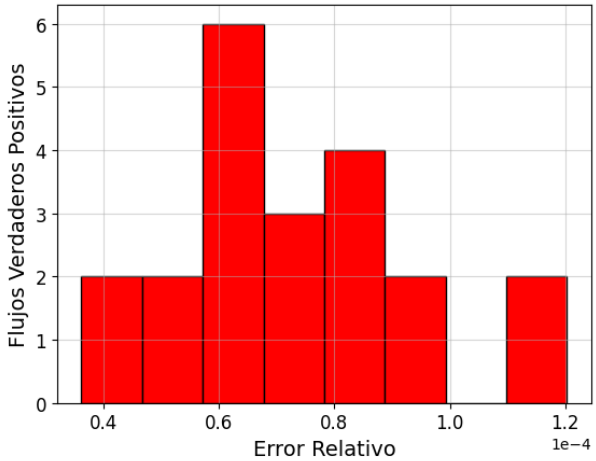
(a) KLLTuple con Top-$K = 15$



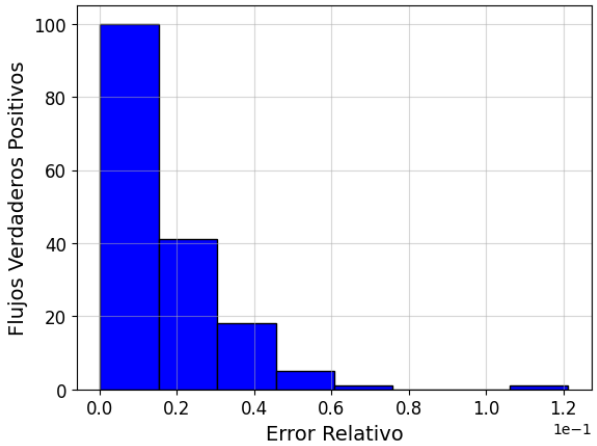
(b) ApacheDatasketch con Top-$K = 15$



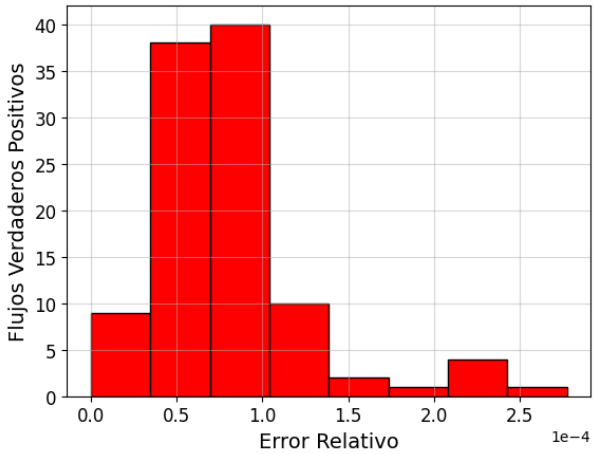
(c) KLLTuple con Top-$K = 63$



(d) ApacheDatasketch con Top-$K = 63$

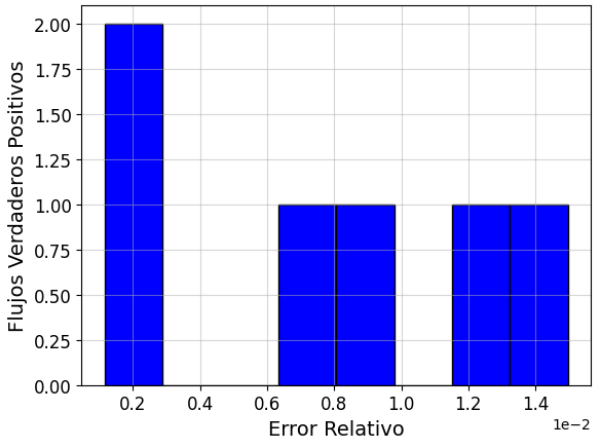


(e) KLLTuple con Top-$K = 255$

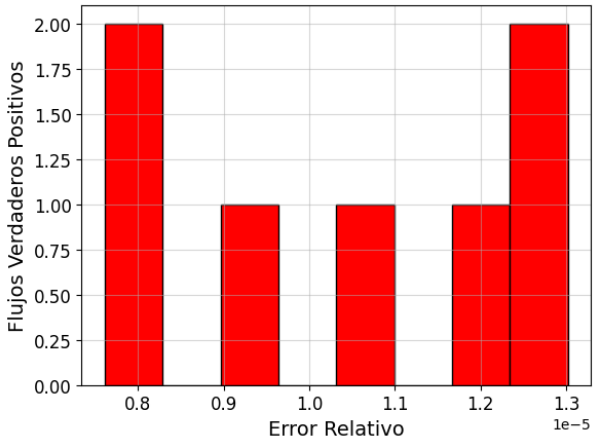


(f) ApacheDatasketch con Top-$K = 255$

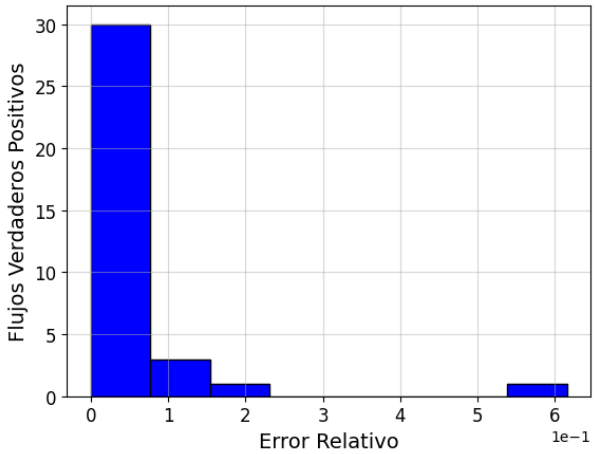
Figura 4.5.2: Histograma que indica la cantidad de flujos identificados como verdaderos positivos según el error relativo obtenido para los sketches KLLTuple y ApacheDatasketch con $\varepsilon = 0{,}003$ en el dataset flujosChicago2016.



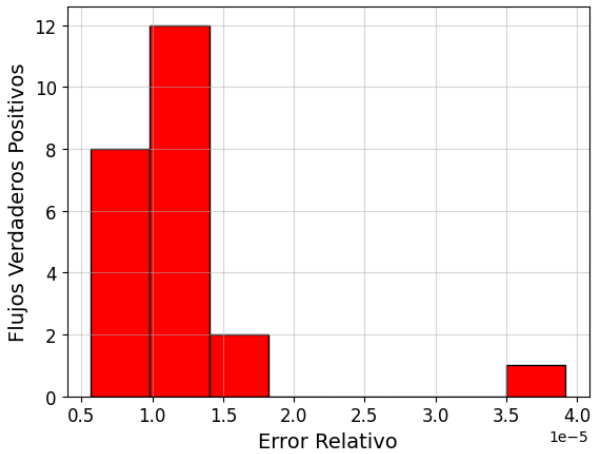
(a) KLLTuple con Top-$K = 15$



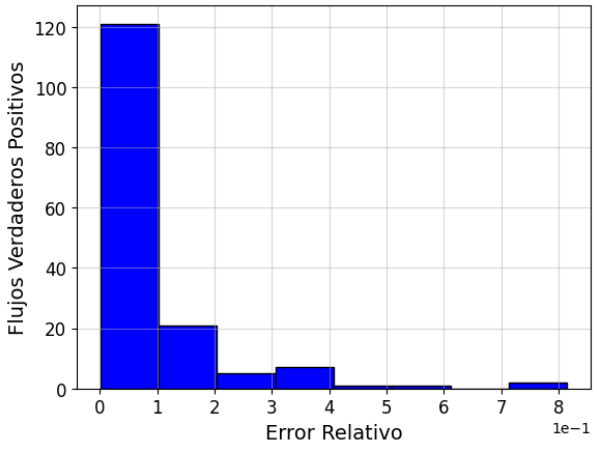
(b) ApacheDatasketch con Top-$K = 15$



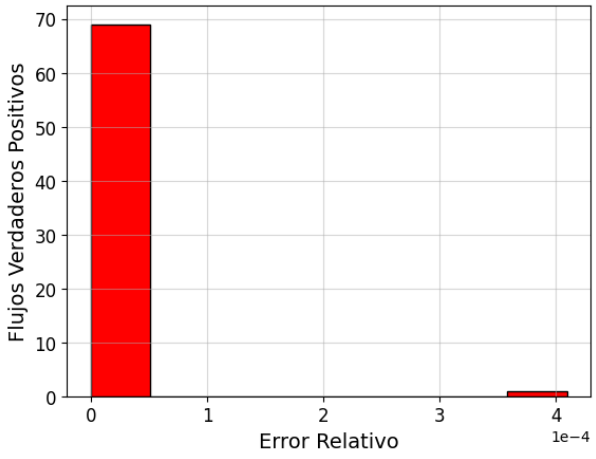
(c) KLLTuple con Top-$K = 63$



(d) ApacheDatasketch con Top-$K = 63$



(e) KLLTuple con Top-$K = 255$



(f) ApacheDatasketch con Top-$K = 255$

Figura 4.5.3: Histograma que indica la cantidad de flujos identificados como verdaderos positivos según el error relativo obtenido para los sketches KLLTuple y ApacheDatasketch con $\varepsilon = 0{,}003$ en el dataset flujosMawi2016.

Capítulo 5

Conclusión

Los sketches KLL y MRL implementados permiten estimar el rank de las frecuencias de las IP encontradas en el monitoreo del tráfico de la red. Sin embargo, los resultados obtenidos indican que el sketch KLL proveniente de la biblioteca ApacheDatasketch [23] obtiene un error absoluto normalizado de aproximadamente un orden de magnitud menor en comparación al KLL implementado, el cual emplea una cantidad similar de espacio. Por otro lado, el MRL obtuvo resultados comparables con ApacheDatasketch al coste de incorporar una cantidad significativa de espacio. También se revisa como estos sketches pueden hacer la operación select, donde se llega a conclusiones similares a las obtenidas para rank.

Se logra realizar una implementación de una variante del sketch KLL, denominada KLLTuple, que permite estimar los top-K flujos que contienen un mayor monitoreo de la red. Adicionalmente, se modifica el sketch proveniente de la biblioteca para que también soporte las consultas de este tipo. A partir de ello, los resultados indican que el sketch KLLTuple obtiene mayores valores de precisión comparados con ApacheDatasketch, con valores tan altos como 0.67 y 0.43 respectivamente. No obstante, este último sketch tiene un error relativo despreciable al momento de estimar el valor del payload de los flujos almacenados, a diferencia del KLLTuple que obtiene resultados con un error relativo de hasta 0.06.

5.1. Trabajo Futuro

Existen distintos lugares de mejora en el sketch KLL implementado, los cuales pueden incrementar el rendimiento de la estructura. Una de las optimizaciones proviene de que el sketch requiere del procesamiento de la mayoría de los datos para poder proporcionar resultados que aprovechen la totalidad del espacio empleado, lo que ocurre debido a que el KLL implementado utiliza una tasa de muestreo alta desde un inicio. Para combatir esta problemática se puede ir cambiando el valor de la tasa de muestreo a medida que se procesan los datos provenientes de un stream. Un ejemplo de ello se encuentra en el paper del sketch KLL [8] que emplea un número constante de compactores con una tasa de muestreo inicial de 1. Cuando el compactor de mayor nivel es compactado, cada compactor aumenta su nivel en 1, por lo

que se duplica el valor de la tasa de muestreo para reflejar el peso asociado a los elementos de cada compactor.

Otra optimización consiste en el uso de compactores como es mencionado por los autores en [24], en el que se proponen varias modificaciones a la compactación y, además, se emplea memoria compartida que busca reducir el tiempo y número de compactaciones realizadas por el sketch. Los cambios propuestos en la compactación consisten en implementar diversas estrategias que modifican cómo se seleccionan los elementos a compactar. Dichas modificaciones buscan mejorar los resultados prácticos, pues poseen las mismas cotas teóricas que el sketch propuesto en el KLL [8].

Queda por investigar cuál será el impacto de incorporar las modificaciones mencionadas al rendimiento del sketch con tuplas. Además, falta verificar cómo afecta la incorporación de la reducción a las operaciones rank y select, esto debido a que al efectuar la unión de dos flujos aumenta el payload asociado y, por ende, modifica los valores percibidos por el sketch. Un ejemplo de ello consiste en que dentro del sketch pueden existir flujos cuyo payload asociado corresponda a 1000, pero que dicho valor no exista dentro del dataset utilizado. Finalmente queda comparar cómo se podría beneficiar la inclusión de una técnica de muestreo al sketch con tuplas proveniente de la biblioteca ApacheDatasketch, ya que los valores de precisión obtenidos en el KLL implementado aumentaban al tener una cota de error tolerado menos estricta, lo que resulta en tasas de muestreo más altas.

Referencias

[1] Qiao Xiao, Xicheng Cai, Ying Qin, Zhan Tang, Shunxi Chen, and Yuxuan Liu. Universal and accurate sketch for estimating heavy hitters and moments in data streams. IEEE/ACM Transactions on Networking, 2023.

[2] Yaime Jiménez. Aceleración hardware de medición de tráfico de red para detección de anomalías. Technical report, 2022.

[3] Jesús Galeano-Brajones, Javier Carmona-Murillo, Juan F Valenzuela-Valdés, and Francisco Luna-Valero. Detection and mitigation of dos and ddos attacks in iot-based stateful sdn: An experimental approach. Sensors, 20(3):816, 2020.

[4] Haoran Han, Zhen Yan, Xuyang Jing, and Witold Pedrycz. Applications of sketches in network traffic measurement: A survey. Information Fusion, 82:58–85, 2022.

[5] Philippe Flajolet, Éric Fusy, Olivier Gandouet, and Frédéric Meunier. Hyperloglog: The analysis of a near-optimal cardinality estimation algorithm. 2007.

[6] Graham Cormode and Muthukrishnan Muthukrishnan. Approximating data with the count-min sketch. IEEE Software, 29(1):64–69, 2012.

[7] Jelani Nelson. Sketching algorithms. Technical report, 2020.

[8] Zohar Karnin, Kevin Lang, and Edo Liberty. Optimal quantile approximation in streams. 2016.

[9] Liang Wang, Gang Luo, Ke Yi, and Graham Cormode. Quantiles over data streams: An experimental study. In Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data, pages 737–748, 2013.

[10] Feng Zhao, Sreenath Maiyya, Ryan Wiener, Divyakant Agrawal, and Amr El Abbadi. Kll$^{\pm}$ approximate quantile sketches over dynamic datasets. 2021.

[11] Yang Zhou, Peng Liu, Hao Jin, Tong Yang, Shoujiang Dang, and Xiaoming Li. One memory access sketch: A more accurate and faster sketch for per-flow measurement. In GLOBECOM 2017 - IEEE Global Communications Conference, pages 1–6, 2017.

[12] Marianne Durand and Philippe Flajolet. Loglog counting of large cardinalities. In European Symposium on Algorithms, pages 605–617, 2003.

[13] Xuyang Jing, Hui Han, Zheng Yan, and Witold Pedrycz. Supersketch: A multi-dimensional reversible data structure for super host identification. IEEE Transactions on Dependable and Secure Computing, 2021.

[14] Amit Goyal, Hal Daumé III, and Graham Cormode. Sketch algorithms for estimating point queries in nlp. In Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning, pages 1093–1103, 2012.

[15] Graham Cormode and Marios Hadjieleftheriou. Finding frequent items in data streams. Proceedings of the VLDB Endowment, 1(2):1530–1541, 2008.

[16] Quartiles and quantiles, n.d.

[17] J Ian Munro and Mike S Paterson. Selection and sorting with limited storage. Theoretical Computer Science, 12(3):315–323, 1980.

[18] Gurmeet Singh Manku, Sridhar Rajagopalan, and Bruce G. Lindsay. Random sampling techniques for space efficient online computation of order statistics of large datasets. In SIGMOD Record, volume 28, pages 251–262, 1999.

[19] A review of recent results in streaming quantiles. a reading project for mit 6.854. Technical report, n.d.

[20] Petar D. Bojovic, Ilija Basicevic, Stanislav Očovaj, and Miroslav Popovic. Ddos attack scoreboard dataset. Technical report, 2017. Data retrieved from Mendeley.

[21] CAIDA. The caida ucsd anonymized internet traces. Technical report, 2008. Data retrieved from CAIDA.

[22] Romain Fontugne, Pierre Borgnat, Patrice Abry, and Kensuke Fukuda. Mawilab: Combining diverse anomaly detectors for automated anomaly labeling and performance benchmarking. In Proceedings of the 6th International Conference, pages 1–12, 2010.

[23] Apache DataSketches. Community, n.d.

[24] Nikolai Ivkin, Edo Liberty, Kevin Lang, Zohar Karning, and Vladimir Braverman. Streaming quantiles algorithms with small space and update time. 2022.