Detección de CNV

Análisis Genómico Avanzado desde Archivos BAM

Integrantes:

- Diego Oyarzo
- Gabriel Huerta

DESAFÍO Y SOLUCIÓN: Eficiencia en Almacenamiento y Memoria

Problema: Limitación de Almacenamiento

Según el National Human Genome Research Institute (NHGRI) y el Human Genome Project (Nature, 2001), el genoma humano tiene aproximadamente 3 mil millones de pares de bases, la detección de CNVs requiere mucha precisión, entonces se usan lecturas de 300 veces el tamaño de bases (300x). Almacenar y procesar todos estos valores de forma directa conlleva un alto consumo de memoria y tiempo de cómputo.

La Solución: Quantile Sketches (KLL)

Proponemos el uso de una solucion con KLL (K-Largest Link) sketches, permitiendo construir un resumen de la distribución de las coberturas de todos los bins en un espacio de memoria constante, independientemente del número total de bins. Lo que nos otorga una solucion en forma de analisis preventivo, mas no como un reemplazo completo, para la deteccion de CNVs.

Necesitamos una representación compacta, eficiente y precisa para analizar la distribución de coberturas por cromosoma.

“Benchmarking of Germline Copy Number Variant Callers from Whole Genome Sequencing Data for Clinical Applications.”Autores: Francisco M. De La Vega, Sean A. Irvine, Pavana Anur, Kelly Potts, Lewis Kraft, Raul Torres, Peter Kang, Sean Truong, Yeonghun Lee, Shunhua Han, Vitor Onuchic, James Han.

Publicado en: medRxiv (preprint).Año: 2024.DOI: https://doi.org/10.1101/2024.07.12.24310338

Origen Archivo .bam

- FASTQ: https://ftp-trace.ncbi.nlm.nih.gov/giab/ftp/data/AshkenazimTrio/HG002_NA24385_son/NIST_HiSeq_HG002_Homogeneity-10953946/HG002_HiSeq300x_fastq/
- FASTA: http://hgdownload.soe.ucsc.edu/goldenPath/hg38/bigZips/hg38.fa.gz (GRCh38)
- Bam generado por NIST usando NovoAlign en 2015 como parte del proyecto Genome in a Bottle.

Lectura y Procesamiento Inicial del Archivo BAM

Apertura del Archivo BAM

Se abre el archivo BAM con HTSlib. La función `sam_open` devuelve un puntero para operaciones posteriores. El header es crucial para interpretar los metadatos del BAM.

```cpp
#include <iostream>
#include <iomanip>
#include <unordered_map>
#include <string>
#include <chrono>

#include <htslib/sam.h>
#include "datasketches-cpp/kll/include/kll_sketch.hpp"

using namespace datasketches;

int main(int argc, char* argv[]) {

    if (argc != 3) {
        std::cerr << "Uso: " << argv[0] << " <archivo.bam> <bin_size>\n";
        return 1;
    }

    const char* bam_file = argv[1];
    int bin_size = std::atoi(argv[2]);

    std::cout << "Analizando cobertura en bins de "
              << bin_size << " bp\n";
```

```cpp
    // --- Abrir BAM ---
    samFile* bam_fp = sam_open(bam_file, "r");
    if (!bam_fp) {
        std::cerr << "Error al abrir BAM\n";
        return 1;
    }

    sam_hdr_t* header = sam_hdr_read(bam_fp);
    if (!header) {
        std::cerr << "Error leyendo header\n";
        sam_close(bam_fp);
        return 1;
    }
```

Optimización de Memoria y Rendimiento

```cpp
// --- Alineamientos ---
bam1_t* aln = bam_init1();

// --- KLL Sketch ---
kll_sketch<float> coverage_sketch(200);
using clock = std::chrono::steady_clock;
std::chrono::duration<double> kll_time{0};

// --- Estado por cromosoma ---
std::string current_chr = "";
std::unordered_map<uint32_t, uint32_t> bins;
bins.reserve(100000); // evita rehash

uint64_t total_reads = 0;
```

Inicialización de Estructuras

Se inicializa `bam1_t` para cada alineamiento y se crea un sketch KLL con `k determinado` para alimentación incremental. `current_chr` rastrea el cromosoma actual.

Mapeo Eficiente de Bins

Un `unordered_map` mapea números de bin a coberturas usando `uint32_t` como claves, lo que es más rápido que usar strings.

Pre-asignación de Memoria

Se pre-asigna espacio para 100k bins (`reserve(100000)`), evitando costosos rehashing y manteniendo operaciones O(1).

Gestión Constante de Memoria

La memoria es constante y proporcional al cromosoma más grande, procesando y liberando datos continuamente. Esto permite analizar genomas grandes sin agotar la RAM.

```cpp
// Leer BAM y contar cobertura por bin
while (sam_read1(bam_fp, header, aln) >= 0) {

    total_reads++;

    //cada 5 millones de reads procesados, imprimir progreso
    if (total_reads % 100'000'000 == 0)
        std::cout << "Procesados: " << total_reads << "\n";

    // Filtros que saltan reads que no deben contar para cobertura
    // Donde BAM_FUNMAP = read no mapeado
    //       BAM_FSECONDARY = read secundario
    //       BAM_FSUPPLEMENTARY = read suplementario
    //       BAM_FDUP = read duplicado
    if (aln->core.flag & (BAM_FUNMAP |
                          BAM_FSECONDARY |
                          BAM_FSUPPLEMENTARY |
                          BAM_FDUP))
        continue;

    // Obtener cromosoma actual
    std::string chr = header->target_name[aln->core.tid];

    // Inicialización
    if (current_chr.empty())
        current_chr = chr;

    // Cambio de cromosoma → flush
    if (chr != current_chr) {

        auto t1 = clock::now();
        for (const auto& e : bins)
            coverage_sketch.update(static_cast<float>(e.second));
        auto t2 = clock::now();

        kll_time += (t2 - t1);


        bins.clear();
        current_chr = chr;
    }

    int start = aln->core.pos;
    int end   = bam_endpos(aln);

    //Recorrer posiciones del read y actualizar bins
    for (int p = start; p < end; p += bin_size) {
        uint32_t bin = p / bin_size;
        bins[bin]++;
    }
}
```

Filtrado de Reads y Detección de Cromosomas

1

Lectura de Alineamientos

`sam_read1` lee el siguiente alineamiento. Un valor `>= 0` indica éxito; `-1` es fin de archivo o error. Se incrementa el contador de reads totales.

2

Filtrado de reads

Ej: ignoran reads con la flag `BAM_FDUP` (duplicados de PCR/ópticos) para evitar inflar artificialmente la cobertura. Esto es crucial para análisis de variantes y CNVs.

3

Detección de Cambio de Cromosoma

Cuando el cromosoma actual difiere del anterior, se procesan los bins acumulados, se insertan en el sketch KLL y se libera la memoria de los bins anteriores.

Cálculo de Cobertura por Bin

4

Se extraen las posiciones de inicio y fin del read. El bucle itera sobre las posiciones cubiertas por el read en saltos de `bin_size`.

Para cada posición, se calcula el número del bin con división entera `p / bin_size`. Por ejemplo, si `bin_size` es 1000, las posiciones 0-999 dan bin 0, 1000-1999 dan bin 1, etc.

Resultados y Limpieza Final

Procesamiento Final del Cromosoma

Se aplica el mismo flujo para el cromosoma final que no fue incluido en el bucle principal. Se libera la memoria del alineamiento, se destruye el header y se cierra el archivo BAM.

```
// Flush final
auto t1 = clock::now();
for (const auto& e : bins)
    coverage_sketch.update(static_cast<float>(e.second));
auto t2 = clock::now();

kll_time += (t2 - t1);


// Limpieza
bam_destroy1(aln);
sam_hdr_destroy(header);
sam_close(bam_fp);
```

Calculo percentiles

Se calculan algunos percentiles para ser usados mas adelantes.

```
float p1  = coverage_sketch.get_quantile(0.01);
float p5  = coverage_sketch.get_quantile(0.05);
float p25 = coverage_sketch.get_quantile(0.25);
float p50 = coverage_sketch.get_quantile(0.50);
float p75 = coverage_sketch.get_quantile(0.75);
float p95 = coverage_sketch.get_quantile(0.95);
float p99 = coverage_sketch.get_quantile(0.99);

std::cout << "P1  (percentil 1):  " << p1  << "x\n";
std::cout << "P5  (percentil 5):  " << p5  << "x\n";
std::cout << "P25 (Q1):           " << p25 << "x\n";
std::cout << "P50 (Mediana):      " << p50 << "x  ← Cobertura típica\n";
std::cout << "P75 (Q3):           " << p75 << "x\n";
std::cout << "P95:                " << p95 << "x\n";
std::cout << "P99:                " << p99 << "x\n";
std::cout << "Mínimo:             " << coverage_sketch.get_min_item() << "x\n";
std::cout << "Máximo:             " << coverage_sketch.get_max_item() << "x\n";
```

```cpp
// Umbrales basados en la mediana
float deletion_threshold    = p50 * 0.5f;
float duplication_threshold = p50 * 1.5f;

std::cout << "Cobertura normal (IQR): "
          << p25 << "× - " << p75 << "×\n";
std::cout << "Umbral deleción (< 0.5× mediana): "
          << deletion_threshold << "×\n";
std::cout << "Umbral duplicación (> 1.5× mediana): "
          << duplication_threshold << "×\n";

// --- Métricas de compresión ---
std::cout << "\n=== MÉTRICAS DE COMPRESIÓN ===\n";
std::cout << "Items retenidos en KLL: "
          << coverage_sketch.get_num_retained() << "\n";
std::cout << "Parámetro k: 200\n";

// --- Tiempo KLL ---
std::cout << "\n=== RENDIMIENTO KLL ===\n";
std::cout << "Tiempo total KLL (resumen de cobertura): "
          << kll_time.count() << " segundos\n";

    return 0;
}
```

Explicación y Adaptabilidad de los Umbrales de CNV

Base Heurística de los Umbrales

Los umbrales de 0.5x y 1.5x la cobertura mediana se basan en la expectativa de un genoma diploide. Una deleción heterocigota (pérdida de 1 copia de 2) reduce la cobertura a ~50%, mientras que una duplicación (de 2 a 3 copias) la aumenta a ~150%.

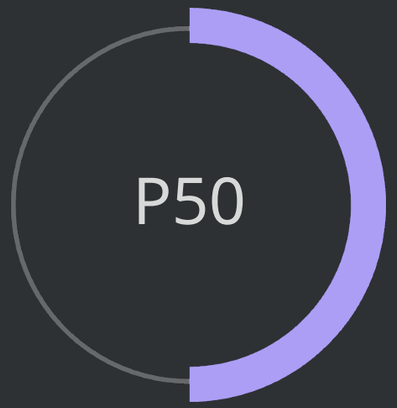
Cálculo Dinámico por Dataset

Es crucial que estos umbrales se calculen dinámicamente para cada conjunto de datos procesado, y no sean valores fijos. Esto asegura que el análisis se adapte a las particularidades de cada muestra, como la profundidad de secuenciación media.

Trazabilidad y Depuración

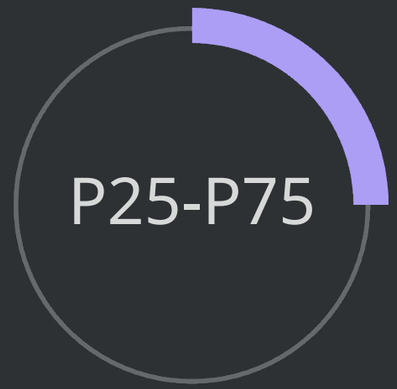
Mostrar los valores exactos de los umbrales de deleción y duplicación (ej., ~1.000× y ~3.000×) es fundamental. Facilita la reproducibilidad de los resultados y permite depurar o validar si un bin fue clasificado como CNV correctamente.

Cuantiles y Umbrales de CNV



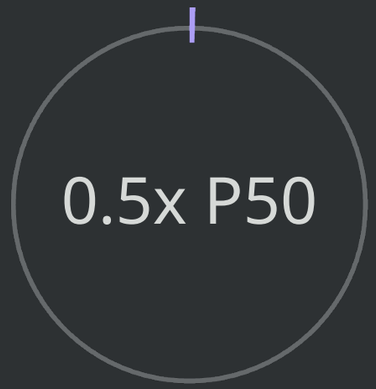
Mediana de Cobertura

El cuantil P50 representa la cobertura mediana, un punto de referencia clave para el análisis.



Rango Normal

Este rango define la "cobertura normal", abarcando el 50% central de los bins.



Umbral de Deleción

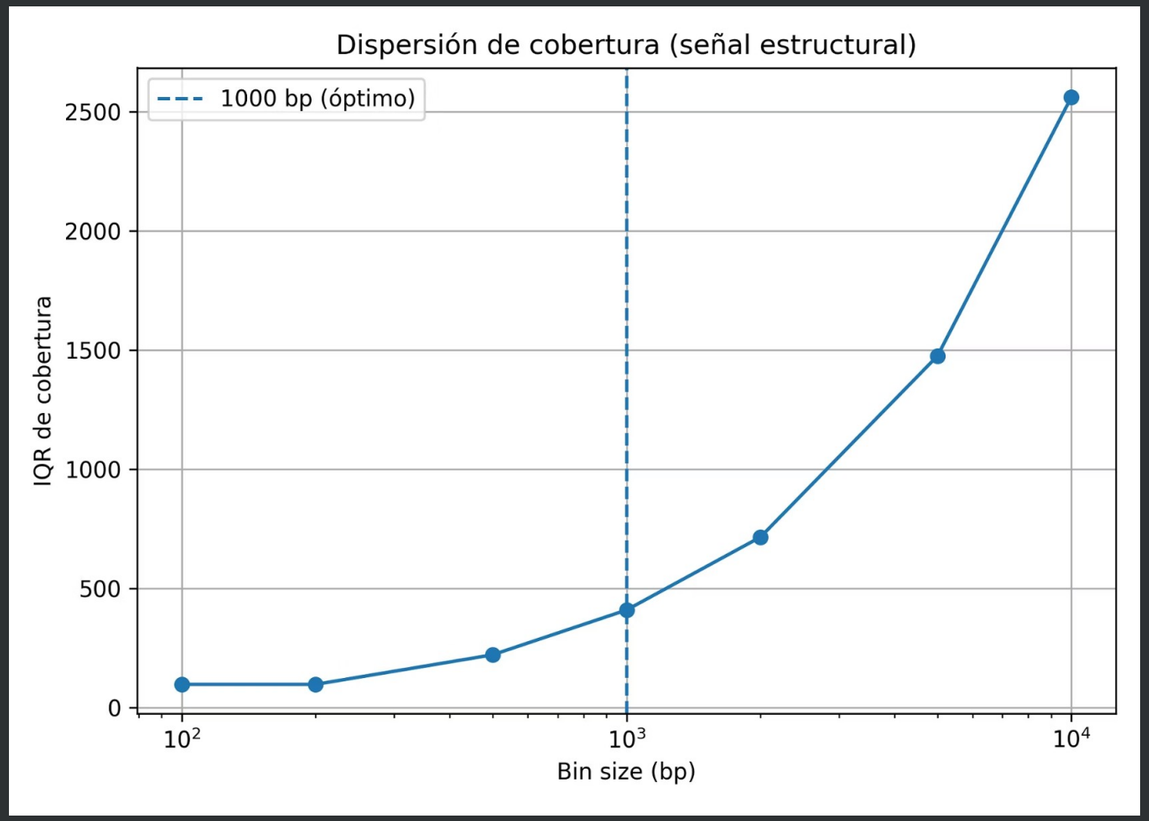
Bins con menos de la mitad de la cobertura mediana (`p50 * 0.5f`) se marcan como posibles deleciones.

Umbral de Duplicación

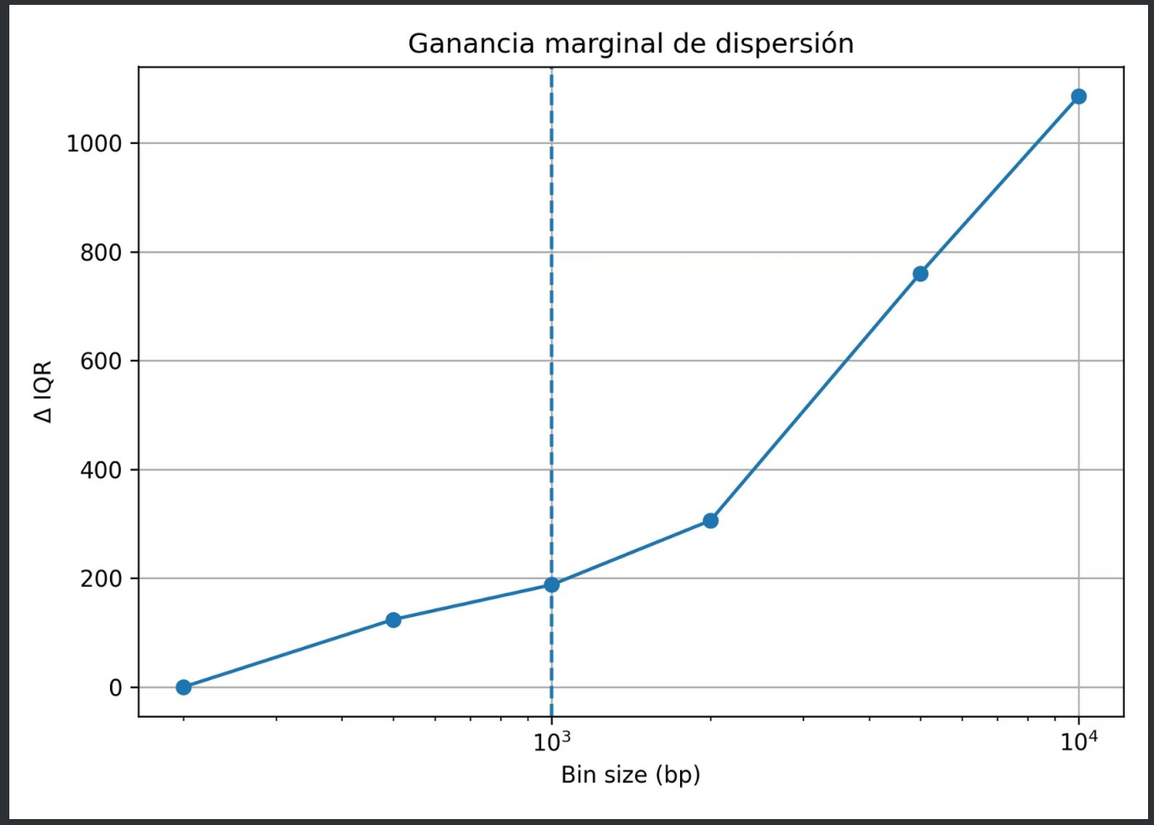
Bins con más de 1.5 veces la mediana (`p50 * 1.5f`) se marcan como posibles duplicaciones.

Experimentación sobre el n° bins

En la experimentación se usó el WGS del genoma humano de referencia GRCh38 con cobertura de 300× (En promedio cada base fue leída unas 300 veces).
Se decidió por dividir el genoma en bins de 1000 bases en base a los siguientes gráficos.



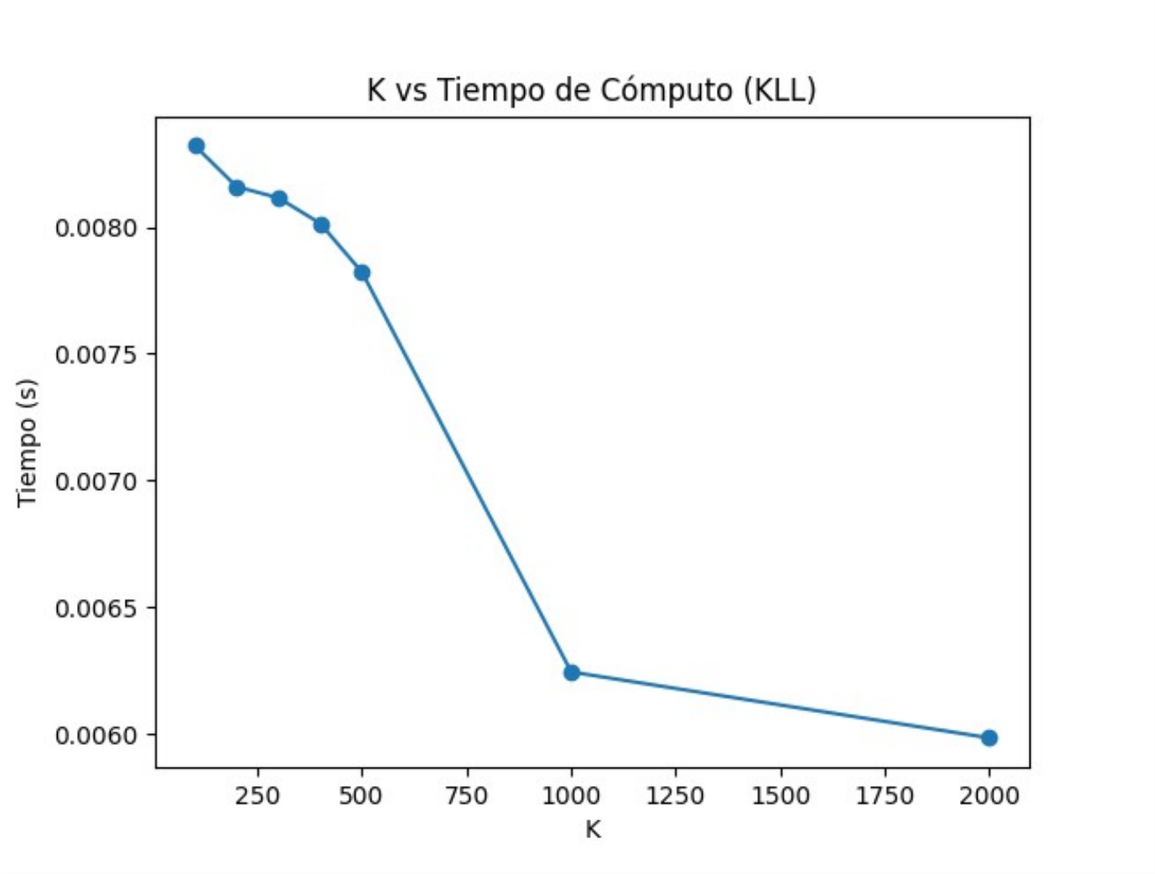
IQR de cobertura indica cuanto varía la cobertura en la mayoría de las regiones (IQR= P75 -P25)

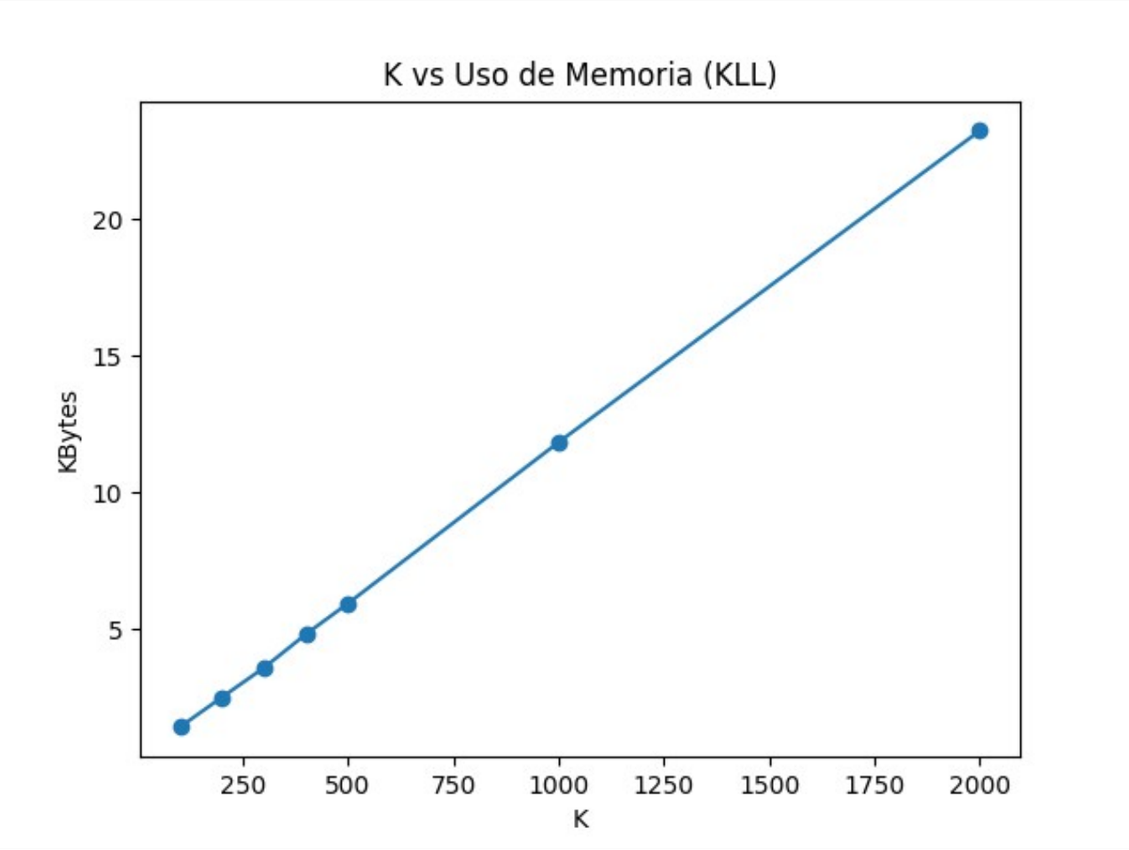


La ganancia marginal de dispersión es una métrica derivada de cuánto aumenta la IQR (IQR_bin_actual - IQR_bin_anterior)

Experimentación sobre el valor k del KLL

Al dividir el genoma en bins de 1000 bases decidimos experimentar sobre como cambia el espacio, tiempo y el error del KLL en base a su valor k (número máximo de elementos por nivel) dando así los siguientes resultados:





Error del KLL sketch

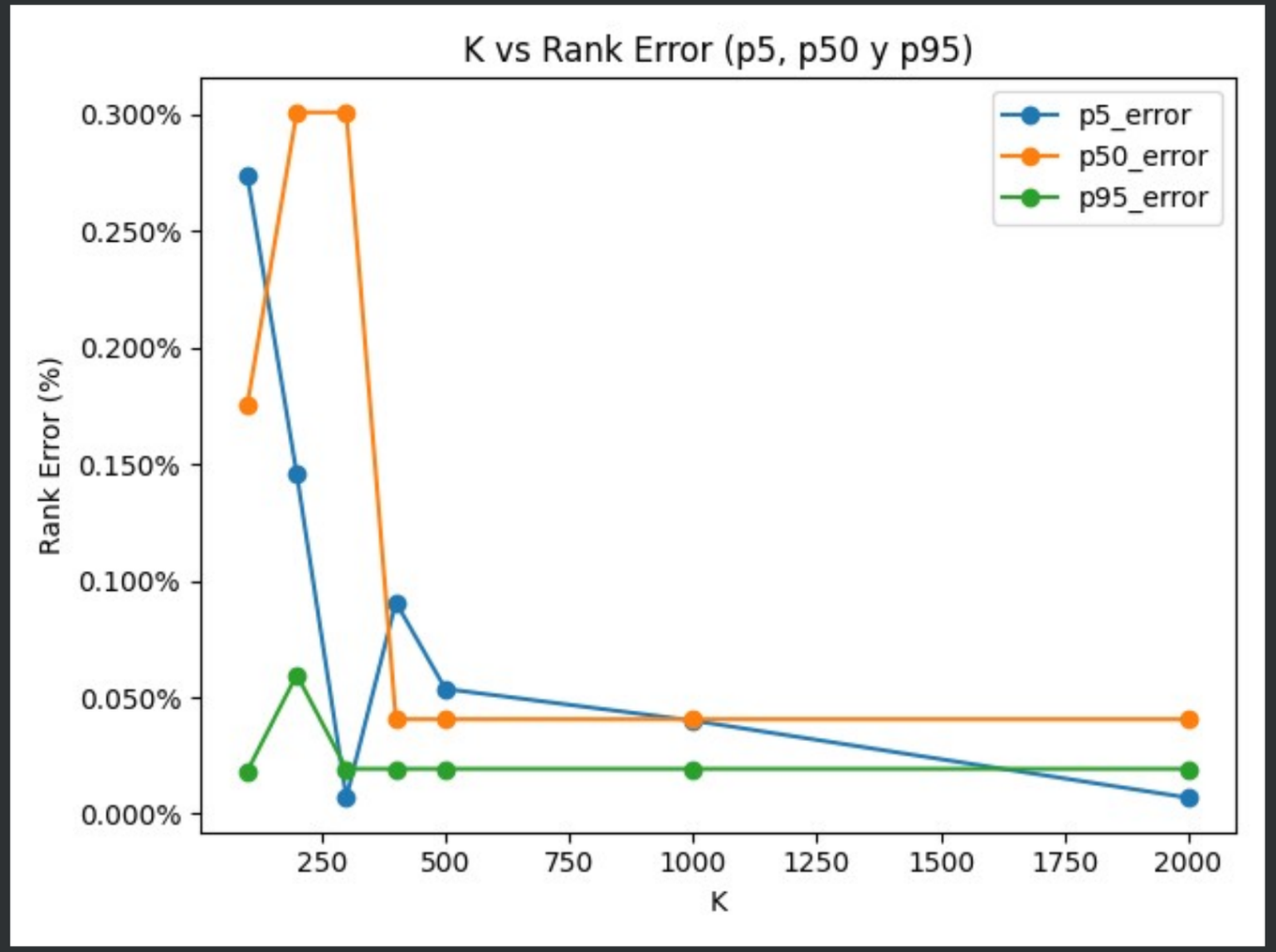
El Rank Error mide la diferencia entre la posición real de un percentil y la posición aproximada entregada por el sketch, normalizada por el tamaño del conjunto de datos

$$\text{RankError}(r) = \left|\hat{r} - r\right| \leq \frac{c}{\sqrt{K}}$$

donde:

- r es el rank verdadero (posición relativa en el conjunto ordenado).
- $\hat{r}$ es el rank estimado por el sketch.
- K es el parámetro del KLL sketch.
- c es una constante del algoritmo (aprox. 1).

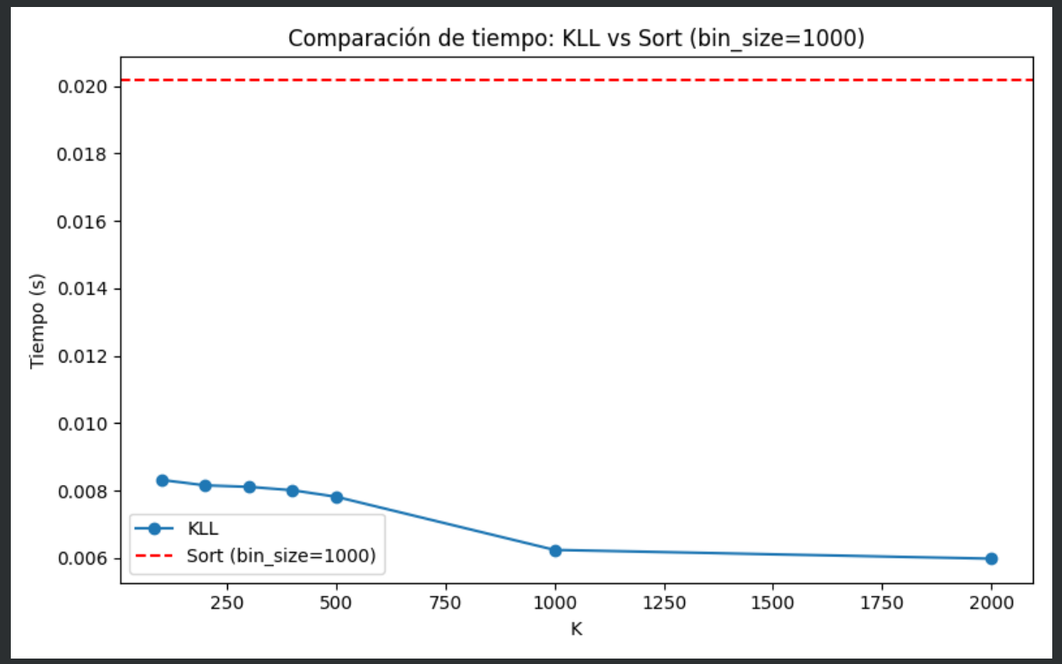
Error del KLL sketch



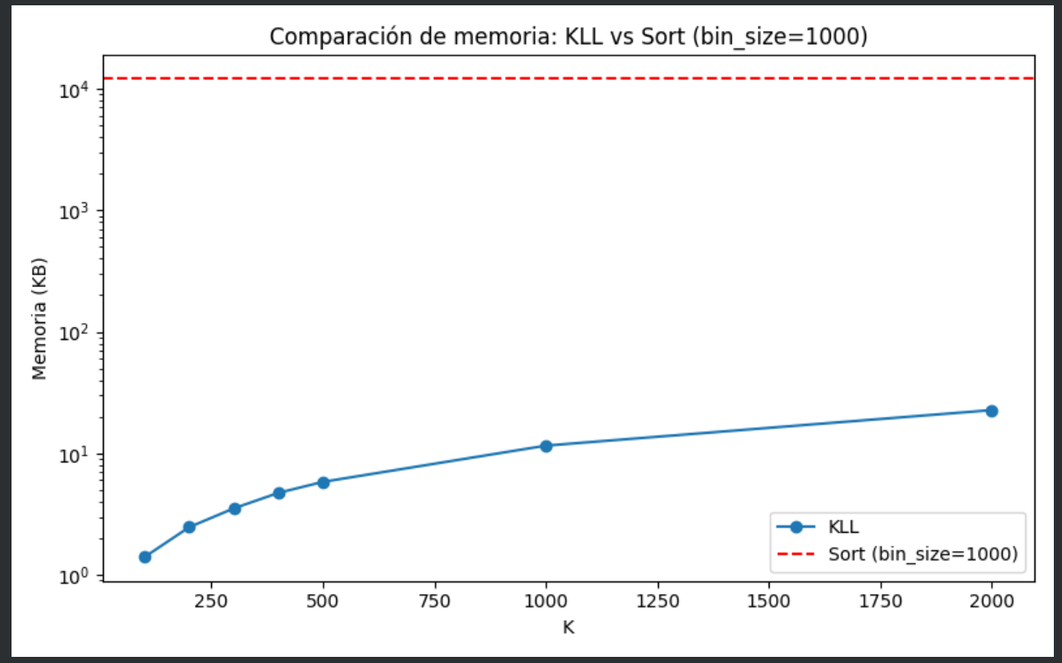
Complejidad Computacional y Eficiencia

Ahora para bins = 1000 y K variable

Tiempo de ejecución



Uso de memoria

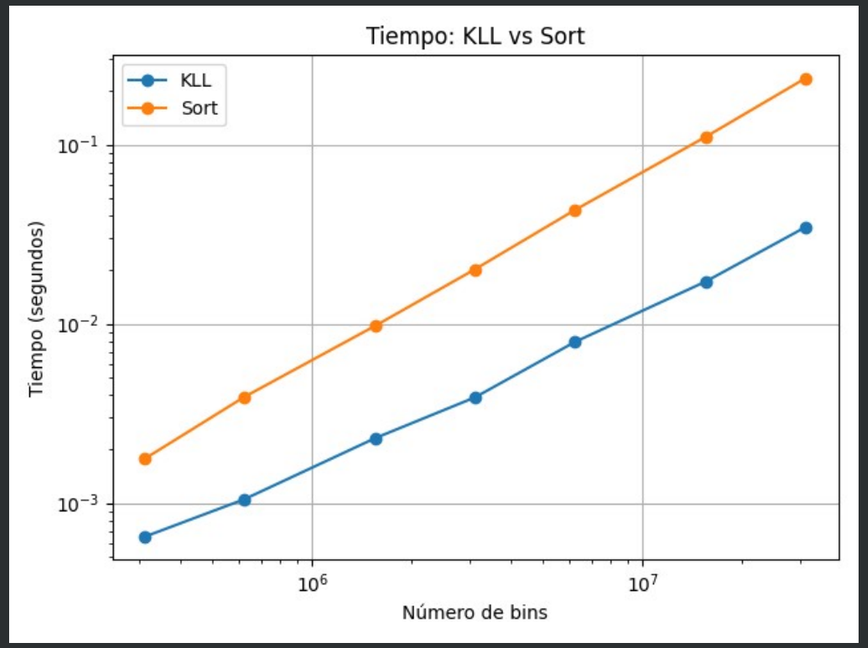


El Kll permite obtener estadísticas robustas de cobertura genómica con memoria constante, mientras que el enfoque basado en sort no escala a genomas completos

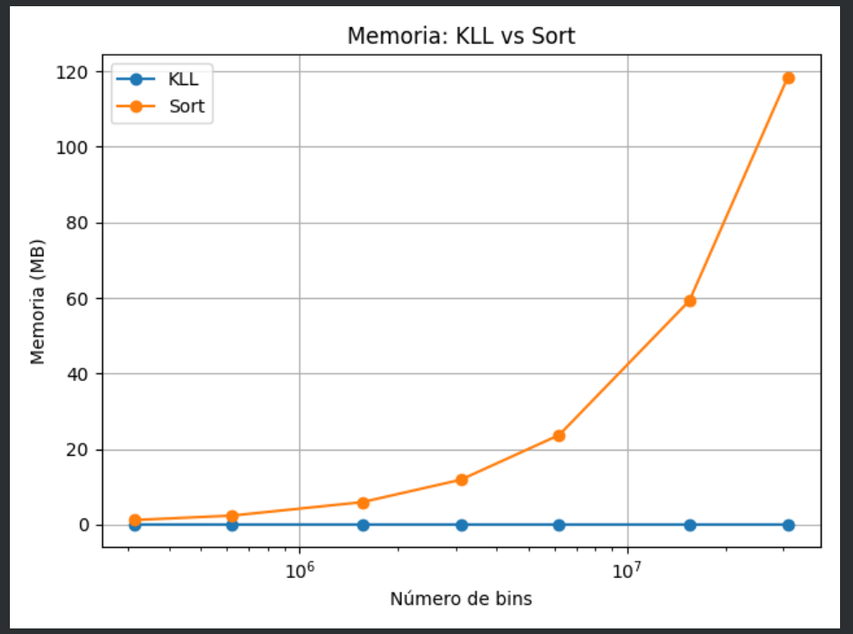
Complejidad Computacional y Eficiencia

Comparación KLL vs Sort para el resumen de cobertura genómica en bins variando el bin size, para k = 4

Tiempo de ejecución



Uso de memoria



Resultados con bins de 1000 bases y K = 400

Se procesó un archivo BAM de 191GB que contiene 5 cromosomas, demostrando eficiencia y precisión en la cuantificación de la cobertura genómica

Fase 1 - Estimación global (KLL)	Fase 2 - Detección local (CNVs)
• Se recorre todo el archivo .bam • Se cuenta las coberturas por bin • Se inserta en el KLL	• Se recorre todo el archivo .bam • Se vuelve a recorrer los bins • Con los umbrales conocidos de la fase 1 se detectan runs de bins anómalos • Guardamos solo CNVs

runs de bins anómalos = Secuencias consecutivas de bins cuyas coberturas sean deleción o duplicación respectivamente.

Estimación global (KLL)

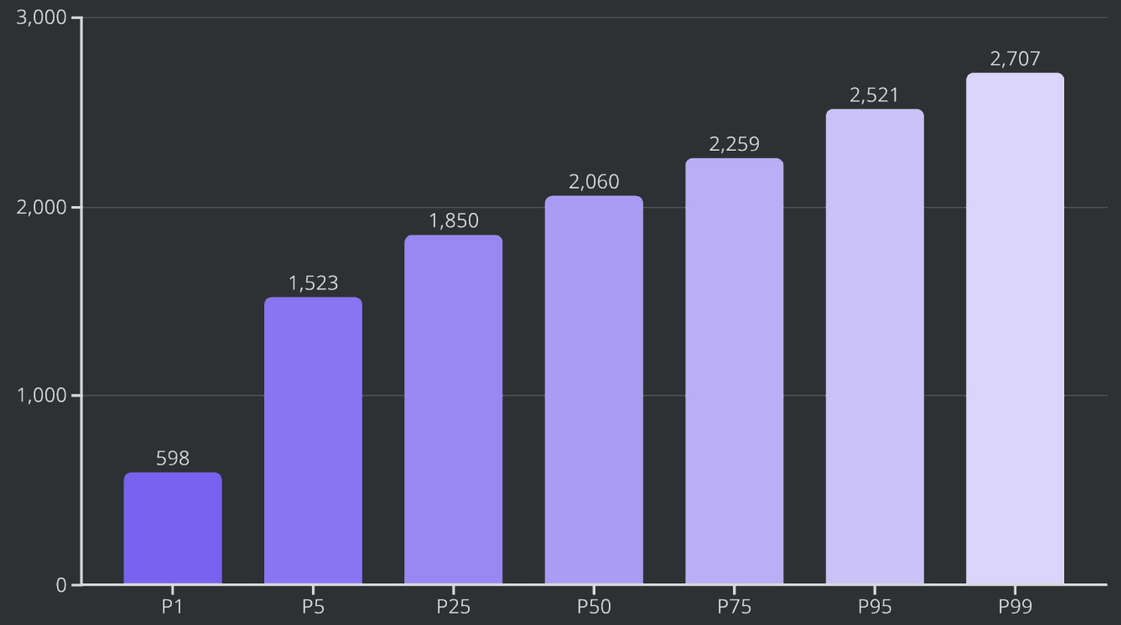
Archivo BAM: 191GB

Total Reads: 3,107,255,000

Bins Analizados: 3,107,255 (tamaño 1kb)

Tiempo de Procesamiento: 15 minutos

Los cuantiles calculados con KLL Sketches revelan la distribución de la cobertura a lo largo del genoma, identificando regiones con variaciones significativas.



Detección local (CNVs)

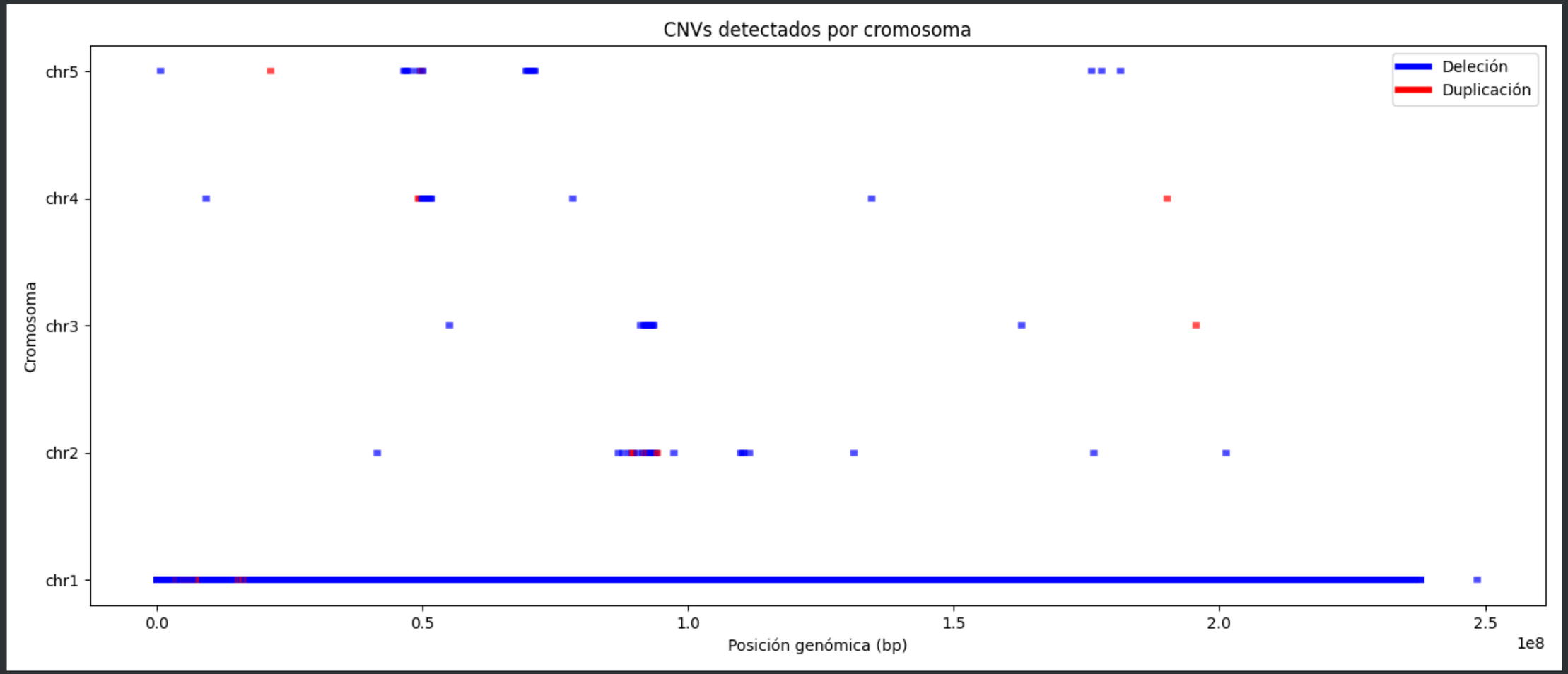
Para runs de bin anómalos >= 1 (de todos los chr)

Deleciones

14070 bins con baja cobertura (< 0.5× mediana).

Duplicaciones

2378 bins con alta cobertura (> 1.5× mediana).



Detección local (CNVs)

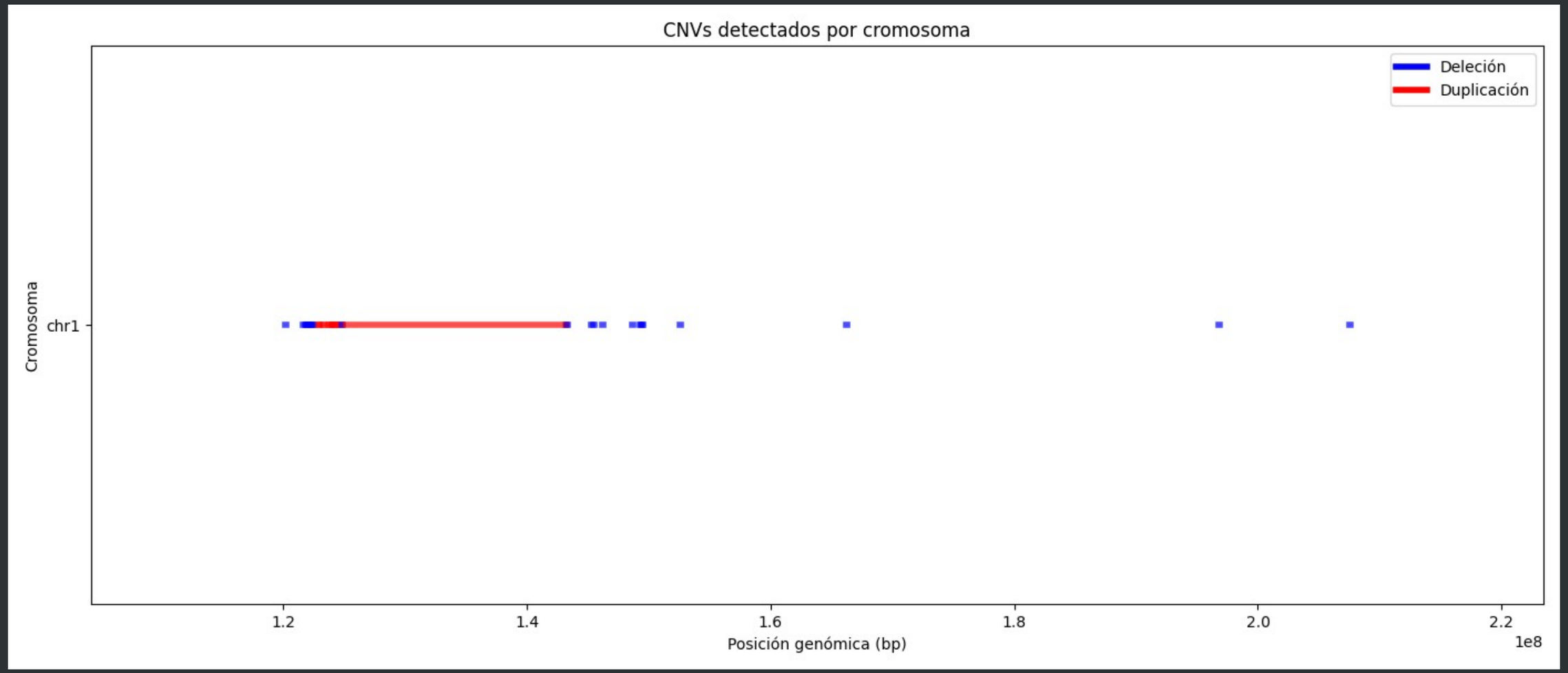
Para runs de bins anómalos >= 5 (todos de chr1)

Deleciones

185 bins con baja cobertura (< 0.5× mediana).

Duplicaciones

294 bins con alta cobertura (> 1.5× mediana).



Conclusiones

- El Sketch KLL permite estimar cuantiles de cobertura de manera eficiente, reduciendo drásticamente el uso de memoria frente a métodos exactos basados en ordenamiento.

- Los resultados experimentales muestran que, con valores moderados de K, es possible obtener errores de rank inferiors al 0.3%, suficientes para identificar desviaciones significativas de cobertura asociadas a CNVs.

- Esto desmuestra que el KLL es una alternativa práctica y escalable para el análisis de grandes volúmenes de datos genómicos.

- Sin embargo desconocemos al 100% su potencial en un archivo WGS complete, debido a que pudimos experimentar con 5 cromosomas y nó con el genoma completo, debido al peso en GB del archivo del genoma (500GB).