IGD: high-performance search for large-scale genomic interval datasets

Jianglin Feng[1,✉] and Nathan C. Sheffield[1-4,✉]

[1]Center for Public Health Genomics, University of Virginia
[2]Department of Public Health Sciences, University of Virginia
[3]Department of Biomedical Engineering, University of Virginia
[4]Department of Biochemistry and Molecular Genetics, University of Virginia
✉ Correspondence: jf.xtable@gmail.com
✉ Correspondence: nsheffield@virginia.edu

Summary: Databases of large-scale genome projects now contain thousands of genomic interval datasets. These data are a critical resource for understanding the function of DNA. However, our ability to examine and integrate interval data of this scale is limited. Here, we introduce the integrated genome database (IGD), a method and tool for searching genome interval datasets more than three orders of magnitude faster than existing approaches, while using only one hundredth of the memory. IGD uses a novel linear binning method that allows us to scale analysis to billions of genomic regions. **Availability**: `https://github.com/databio/IGD`

INTRODUCTION

Genome and epigenome data are frequently represented as *genomic intervals*, defined by a chromosome plus *start* and *end* coordinates. Because this data type is so common, interval-based comparisons are fundamental to genomic analysis[1–4]. The most basic operation on interval data is computing the intersections of two interval sets. This operation is common for a variety of use cases such as determining overlaps between sets of transcription factor binding sites, plotting the genome-wide distribution of a set of SNPs or regions, or assigning genomic annotations to query regulatory elements. Several approaches have been developed to do calculation this efficiently, such as the nested containment list (NCList)[5], the interval tree[6], the R-tree[7], and the Augmented Interval List (AIList)[8], and many tools exploiting these data structures exist[1,9–12].

A more advanced task with intervals is *interval database search*, which requires comparing a query interval set against a database of multiple interval sets. This task extends the file-to-file comparison to a file-to-database comparison. To do this, the file-to-file approaches can be applied in a loop, but this does not take advantage of integrating across interval sets in the database, which can be done by creating a combined index that spans individual interval sets. This is the approach taken by GIGGLE, which is much faster for searching a large database than a naive file-to-file loop[3]. Here, we present a new approach called the *integrated genome database (IGD)*, which uses a novel indexing method called *linear binning*. We show that the IGD approach achieves substantially better performance than all existing approaches.

METHODS

Linear binning. An interval set has two different orders: the order of start values and the order of end values. To effectively process interval data, multi-level binning methods are usually used[2,7,10], where intervals are stored in bins of different sizes that are large enough to contain the intervals. In contrast, the linear binning approach used by IGD is a single-level binning (Fig. 1a). The whole genome is divided into equal-sized bins and intervals are put into bins they intersect. The intervals are not required to fit into the bins; instead, intervals that cross bin boundaries are stored in multiple bins. This duplication introduces two challenges: increase in size of the database, and a possibility of multiple counting. The number of duplicates can be adjusted by the selecting bin size, which therefore adjusts the amount of extra data required for linear binning. In our tests, the database size increases by only 5% to 20% for a bin size of 16,384 (2^{14}), which is much more efficient than multi-level binning or tree-based methods (see results in the next section). The multiple counting challenge, which occurs only when both the query and the data interval cross the same bin boundary, can be neatly solved by setting a rule: if the query crosses the left boundary of the bin, then any region in the bin that also crosses the left boundary will be skipped. In figure 1a, query q_2 intersects both interval $2a$ (which belongs to bin i) and interval $2b$ (bin $i+1$); since q_2 does not cross the left boundary of bin i, region $2a$ will count; but q_2 crosses the left boundary of bin $i+1$, so the intersection with region $2b$ will not count.

With the challenges of multiple counting and database size neatly solved, linear binning makes interval data behave similar to a dataset with a single order. More importantly, since a bin contains all regions that intersect the bin, the processing of intervals in each bin is independent. In contrast, the nodes of the tree-based or multi-level approaches are not independent and queries typically require traversing nodes or bins of different levels.

IGD data structure and query. IGD stores the bin data linearly and maintains a header that contains the location of the bins and the number of IGD data elements in each bin. Each IGD data element contains the index of the original dataset, the original genomic region ($start$ and end coordinates), and a value (signal level). For compatibility to other existing search tools, IGD also supports a data element type without the signal value. Since the region data is indexed linearly, a query to the database simply determines the index of the bin that the query region intersects and then loads only that bin into memory and searches within the bin. If a query crosses any bin boundaries, IGD loads and searches those bins one by one. Compared to file-to-file approaches, IGD does not need to load all the files and does not need to search all regions in every file, making IGD much more efficient (Fig. 1b).

Regions in a bin are presorted by the region $start$, so the search inside a bin can be done by first using a binary search to find the right-most interval (intervals with $start$ larger than or equal to the query end will not intersect the query) then scanning to the left (smaller $start$). This is very efficient because the bin size, *e.g.*, 16,384 bp, is small. Alternatively, the general search algorithm of AIList[8] may be used, but it requires preprocessing to add an extra

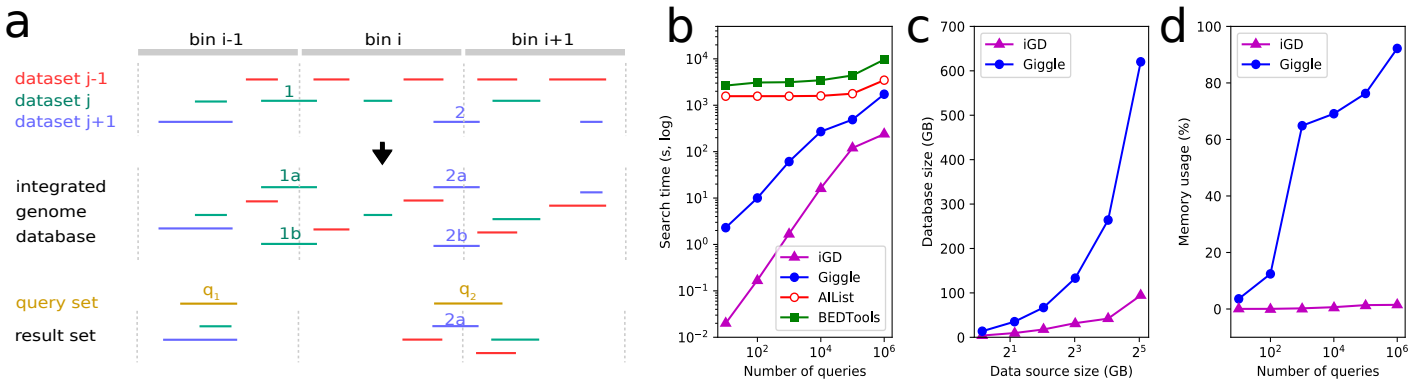

Fig. 1: a) Linear-binning indexing. The genome is divided into equal-sized bins and regions are put into any bins they intersect. Regions that cross boundaries are stored in multiple bins. For example, region 1 is placed in both bin $i-1$ and bin i. Regions in each bin are then sorted by their starts. Search queries may be unsorted. Colors indicate that the file each region belongs to. A search query interval $q2$ crosses the boundary of bin i and bin $i+1$, so region 2 is double counted. IGD solves this by counting only $2a$ (see text); b) Performance comparisons with existing methods on 5660 interval datasets. Search times are shown for IGD, GIGGLE, AIList and BEDTools on 6 query datasets of 10^1 to 10^6 regions; c) Sizes of IGD and GIGGLE databases created from 6 different subsets of a UCSC interval set collection with input size ranging from 2^0 to 2^5 GB of compressed data; d) Peak memory usage for IGD and GIGGLE for the 6 queries. GIGGLE did not complete the query of 10^6 regions.

element (the augmenting value) for each region, which increases the storage by one third and also affects the data loading speed.

RESULTS

To benchmark the speed and memory use of the IGD approach, we assembled a test set of human query and database interval sets. All tests in this work were performed using a single core on a 2.8 GHz personal computer with 16 GB memory and an external 1TB SSD disc. We created the IGD databases with a bin size of 8,192 (2^{13}), corresponding to about 376,000 bins for human genome.

We first compared IGD to existing approaches for the database search task. For query sets with 100 or fewer intervals, IGD is four to five orders of magnitudes faster than file-to-file approaches AIList and BEDTools (Fig 1b). This is expected, as these approaches load all data from 5660 files while IGD only loads data from very few bins. GIGGLE uses a database approach based on B+ tree indexing scheme, which is faster than the file-by-file methods, but IGD is two orders of magnitude faster still. For larger query intervals, IGD is also several orders of magnitude faster than any existing approach.

In terms of disk use, the IGD index increases linearly with the size of input data source, while GIGGLE increases exponentially (Fig 1c). For the full set UCSC data source, GIGGLE uses 640GB disc space while IGD only needs 98GB. IGD is also faster in database creation: the complete database takes IGD about 2 hours to create, while GIGGLE takes more than 4 hours.

Furthermore, IGD uses nearly negligible computer memory while GIGGLE needs substantial memory even for small query sets (Fig. 1d). For example, with 1,000 query regions, IGD requires only 38 MB of memory, while GIGGLE uses 10,570 MB memory, more than 270 times as much. These results suggest that linear-binning indexing is superior to B+tree indexing in terms of database creation time, search time, and search memory use. Bin size is an option set by the user when a database is created. To explore how bin size affects search speed and duplication rate, we compared various datasets with bin sizes ranging from 2^{12} to 2^{18} and found that the default bin size (2^{14}) is near optimal for all scenarios we tested (Supplementary Information). We conclude that for most use cases, the user will not need to worry about bin size.

We have shown how IGD presents a fast alternative for interval search operations. Furthermore, the independence of IGD bins and the small memory use make IGD amenable to parallel processing, which can further increase the searching speed. In additional to calculating number of overlaps, IGD can also calculate the seqpare metric, which provides a novel way to assess similarity between region sets[13]. IGD is well suited for tasks that require searching a database of intervals with a query, such as in region set enrichment overlap calculations[12,14,15], or methods that use collections of region sets to aggregate data[16–19]. As the size of genomic interval data increases and more databases arise to serve and analyze this data type[14,20–22],we believe the speed and memory advances achieved by IGD will make it a valuable tool for fast analysis in the future.

FUNDING

This work was supported by the University of Virginia School of Medicine and the University of Virginia 4-VA program.

REFERENCES

1. Sheffield, N. C. & Bock, C. LOLA: Enrichment analysis for genomic region sets and regulatory elements in R and bioconductor. *Bioinformatics* **32,** 587–589 (2016).

2. Li, X.-Y. *et al.* The role of chromatin accessibility in directing the widespread, overlapping patterns of drosophila transcription factor binding. *Genome Biol.* **12,** R34 (2011).

3. Layer, R. M. *et al.* GIGGLE: A search engine for large-scale integrated genome analysis. *Nature Methods* **15,** 123–126 (2018).

4. Jalili, V., Matteucci, M., Goecks, J., Deldjoo, Y. & Ceri, S. Next generation indexing for genomic intervals. *IEEE Transactions on Knowledge and Data Engineering* 1–1 (2018). doi:10.1109/tkde.2018.2871031

5. Alekseyenko, A. V. & Lee, C. J. Nested containment list (NCList): A new algorithm for accelerating interval query of genome alignment and interval databases. *Bioinformatics* **23,** 1386–1393 (2007).

6. Cormen, T. H., Leiserson, C. E., Rivest, R. L. & Stein, C. Introduction to algorithms second edition. (2001).

7. Kent, W. J. *et al.* The human genome browser at ucsc. *Genome Res.* **12,** 996–1006 (2002).

8. Feng, J., Ratan, A. & Sheffield, N. C. Augmented interval list: A novel data structure for efficient genomic interval search. *Bioinformatics* (2019). doi:10.1093/bioinformatics/btz407

9. Richardson, J. E. Fjoin: Simple and efficient computation of feature overlaps. *Journal of Computational Biology* **13,** 1457–1464 (2006).

10. Quinlan, A. R. & Hall, I. M. BEDTools: A flexible suite of utilities for comparing genomic features. *Bioinformatics* **26,** 841–842 (2010).

11. Neph, S. *et al.* BEDOPS: High-performance genomic feature operations. *Bioinformatics* **28,** 1919–1920 (2012).

12. Dozmorov, M. G. Epigenomic annotation-based interpretation of genomic data: From enrichment analysis to machine learning. *Bioinformatics* **33,** 3323–3330 (2017).

13. Feng, S. C., Sheffield, N. C. & Feng, J. Seqpare: A self-consistent metric of similarity between genomic interval sets. *F1000Research* **9,** 581 (2020).

14. Nagraj, V., Magee, N. & Sheffield, N. C. LOLAweb: A containerized web server for interactive genomic locus overlap enrichment analysis. *Nucleic Acids Research* (2018). doi:10.1093/nar/gky464

15. Kanduri, C., Bock, C., Gundersen, S., Hovig, E. & Sandve, G. K. Colocalization analyses of genomic elements: Approaches, recommendations and challenges. *Bioinformatics* **35,** 1615–1624 (2018).

16. Schep, A. N., Wu, B., Buenrostro, J. D. & Greenleaf, W. J. chromVAR: Inferring transcription-factor-associated accessibility from single-cell epigenomic data. *Nature Methods* **14,** 975–978 (2017).

17. Boer, C. G. de & Regev, A. BROCKMAN: Deciphering variance in epigenomic regulators by k-mer factorization. *BMC Bioinformatics* **19,** (2018).

18. Gomez, L. *et al.* coMethDMR: Accurate identification of co-methylated and differentially methylated regions in epigenome-wide association studies with continuous phenotypes. *Nucleic Acids Research* **47,** e98–e98 (2019).

19. Lawson, J. T., Smith, J. P., Bekiranov, S., Garrett-Bakelman, F. E. & Sheffield, N. C. COCOA: Coordinate covariation analysis of epigenetic heterogeneity. *Genome Biology* **21,** (2020).

20. Yevshin, I., Sharipov, R., Kolmykov, S., Kondrakhin, Y. & Kolpakov, F. GTRD: A database on gene transcription regulation2019 update. *Nucleic Acids Research* **47,** D100–D105 (2018).

21. Oki, S. *et al.* Ch IP -atlas: A data-mining suite powered by full integration of public ch IP -seq data. *EMBO reports* **19,** (2018).

22. Chèneby, J., Gheorghe, M., Artufel, M., Mathelier, A. & Ballester, B. ReMap 2018: An updated atlas of regulatory regions from an integrative analysis of DNA-binding ChIP-seq experiments. *Nucleic Acids Research* **46,** D267–D275 (2017).

Supplementary Information

Authors: Jianglin Feng and Nathan C. Sheffield

1. The impact of bin size on database size and search speed

The choice of the bin-size (l_{bin}) for IGD has greater impact on the size of the created database than on the search speed. Theoretically, this impact depends on the average interval length (l_{avg}) of the interval sets involved. The rate of bins that split the intervals (the duplication rate) is l_{avg}/l_{bin}. Therefore, if l_{bin} is close to l_{avg}, the size of the created IGD database will be twice of the size of the original datasets; to make IGD database size 1/4 larger than the original size, l_{bin} should be $4 \times l_{avg}$. Since IGD uses binary search to determine the overlaps in a bin, $4\times$ changes of l_{bin} only add 2 extra search operations, which has only a minor effect on overall speed.

The following two tables list test results on the database size and search speed for 3 representative collections of interval sets with a wide range of average lengths: roadmap (l_{avg}=7064), a subset of UCSC (l_{avg}=1290), and a subset of JASPAR (l_{avg}=9).

Table S1. Measured duplication ratio ((IGD database size-original size)/original size) of IGD database at different bin-sizes for three different data sources. The calculated duplication ratios are listed in parentheses. For ROADMAP data, the measured values are nearly exact as calculated (l_{avg}/l_{bin}).

bin-size(bp)	ROADMAP	UCSC	JASPAR
4,096	1.725(1.725)	0.310(0.315)	0.003(0.002)
8,192	0.862(0.862)	0.153(0.157)	0.002(0.001)
16,384	0.431(0.431)	0.075(0.079)	0.001(0.001)
32,768	0.216(0.216)	0.035(0.039)	0.0(0.0)
65,536	0.108(0.108)	0.016(0.020)	0.0(0.0)
131,072	0.054(0.054)	0.006(0.010)	0.0(0.0)
262,144	0.026(0.027)	0.004(0.005)	0.0(0.0)

Table S2. Runtime(s) of IGD search for databases created using different bin-sizes (Table S1). The query interval set contains 1 million intervals.

bin-size(bp)	ROADMAP	UCSC	JASPAR
4,096	8.372	20.126	8.668
8,192	5.123	15.556	7.884
16,384	3.806	12.975	7.522
32,768	3.239	12.906	7.290
65,536	3.055	13.664	7.478
131,072	2.890	15.251	7.439
262,144	3.125	18.989	10.34

It can be seen that bin-size of 16384, 32768 and 65536 are generally good for search for databases with regions that range in average size from 9 to 7000+. The default bin-size is set to 16384 (2ˆ14).

2. Performance of IGD on different computing enviroments

The current implementation of IGD is serial, although it has great potential for parallel processing. We tested IGD on both a personal computer and on a high performance cluster (HPC) compute node. IGD runs nearly identically in either case, assuming the hard disk type is the same. With a conventional hard-disk-drive (HDD), results will be similar whether on and on a node of a local cluster when the cluster node has the same processing power as the personal computer. However, we found that IGD runs substantially faster when the database is stored on a solid-state-drive (SSD). With an SSD, the IGD database creation is 3 to 6 times faster and IGD database search is roughly 2 times faster. Therefore, to create an IGD database from large-scale datasets like UCSC (˜100GB), a large volume SSD (1TB or larger) is recommended.