NuchaRt: Embedding High-Level Parallel Computing in R for Augmented Hi-C Data Analysis

Nucha*R*t: embedding high-level parallel computing in R for augmented Hi-C data analysis

Fabio Tordini[1], Ivan Merelli[3], Pietro Liò[2], Luciano Milanesi[3], and Marco Aldinucci[1]

[1] Computer Science Dept, University of Torino, Torino, Italy
{tordini,aldinuc}@di.unito.it
[2] Computer Laboratory, University of Cambridge, Cambridge, UK
pietro.lio@cl.cam.ac.uk
[3] IBT - Italian National Research Council, Segrate (MI), Italy
{luciano.milanesi,ivan.merelli}@itb.cnr.it

Abstract. Recent advances in molecular biology and Bioinformatics techniques brought to an explosion of the information about the spatial organisation of the DNA in the nucleus. High-throughput chromosome conformation capture techniques provide a genome-wide capture of chromatin contacts at unprecedented scales, which permit to identify physical interactions between genetic elements located throughout the human genome. These important studies are hampered by the lack of biologists-friendly software. In this work we present Nucha*R*t, an R package that wraps NuChart-II, an efficient and highly optimized C++ tool for the exploration of Hi-C data. By rising the level of abstraction, Nucha*R*t proposes a high-performance pipeline that allows users to orchestrate analysis and visualisation of multi-omics data, making optimal use of the computing capabilities offered by modern multi-core architectures, combined with the versatile and well known R environment for statistical analysis and data visualisation.

Keywords: Next-Generation Sequencing, Neighbourhood Graph, High-Performance Computing, Multi-Omic Data, Systems Biology

1 Scientific Background

Over the last decade, a number of approaches have been developed to study the organisation of the chromosome at high resolution. These approaches are all based on the Chromosome Conformation Capture (3C) technique, and allow the identification of neighbouring pairs of chromosome loci that are in close enough physical proximity (probably in the range of 10-100 nm) that they become cross-linked [1]. This information highlights the three-dimensional organisation of the chromosome, and reveals that widely separated functional elements actually result to be close to each other, and their interaction can be the key for detecting

critical epigenetics patterns and chromosome translocations involved in the process of genes regulation and expression.

Among 3C-based techniques, the *Hi*-C method exploits Next-Generation Sequencing (NGS) techniques to provide a genome-wide library of coupled DNA fragments that are found to be close to each other in a spatial context. The contact frequency between the two fragments relies on their spatial proximity, and thus it is expected to reflect their distance. The output of a Hi-C process is a list of pairs of locations along all chromosomes *reads*, which can be represented as a square matrix M, where each element $M_{i,j}$ of the matrix indicates the intensity of the interactions between positions i and j.

In a previous work we proposed NuChart-II as a highly optimised, C++ application designed to integrate information about genes positions with paired-ends reads resulting from Hi-C experiments, aimed at describing the chromosome spatial organisation using a gene-centric, graph-based approach [6]. A graph-based representation of the DNA offers a more comprehensive characterization of the chromatin conformation, which can be very useful to create a representation on which other *omics* data can be mapped and characterize different spatially-associated domains.

NuChart-II has been designed using high-level parallel programming patterns, that facilitate the implementation of the algorithms employed over the graph: this choice permits to boost performances while conducting genome-wide analysis of the DNA. Furthermore, the coupled usage of C++ with advanced techniques of parallel computing (such as lock-free algorithms and memory-affinity) strengthens genomic research, because it makes possible to process much faster, much more data: informative results can be achieved to an unprecedented degree [3].

However, C++ is not widely used in Bioinformatics, because it requires highly specialised skills and does not fully support the rapid development of new interactive pipelines. Conversely, the modularity of R and the huge amount of already existing statistical packages facilitates the integration of exploratory data analysis and permits to easily move through the steps of model development, from data analysis to implementation and visualisation. In this article we discuss the integration of our C++ application into the R environment, an important step toward our objective of augmenting the usability of bioinformatics tools: we aim at obtaining a high-performance pipeline that allows users to orchestrate analysis and visualisation of multi-omics data, making optimal use of the computing capabilities offered by modern multi-core architectures, combined with the versatile and well known R environment for statistical analysis and data visualisation. The novel package has been renamed Nucha*R*t.

1.1 Parallelism facilities in R

By default, it is not possible to take advantage of multiple processing elements from within the R environment. Instead, a sort of "back-end" must be registered, that effectively permits to run a portion of code in parallel. For what it concerns high-performance computing, some libraries exist that foster parallel

programming in R, most of which focus on distributed architectures and clusters of computers. Worth to mention are *Rmpi* and *Snow*.

Rmpi is a wrapper to MPI and exposes an R interface to low-level MPI functions. The package provides several R-specific functions, beside wrapping the MPI API: parallel versions of the `apply()`-like functions, scripts to launch R instances at the slaves from the master and some error-handling functions to report errors form the workers to the manager. Snow (Simple Network Of Workstations) provides support for simple parallel computing on a network of workstations and supports several different low-level communication mechanisms, including private virtual machine (PVM), MPI (via Rmpi) and raw sockets. The package also provides high-level parallel functions like `apply()` and simple error-handling mechanism.

The *multicore* package builds a back-end for parallel execution of R code on machines with multiple CPUs: all jobs share the full state of R when parallel instances are spawned, so no data or code needs to be copied or initialized. Spawning uses the `fork` system call (or OS-specific equivalent) and establishes a `pipe` between master and child process, to enable inter-process communication. However, the variety of operations that can be parallelized with *multicore* is limited to simple independent math computations on a collection of indexed data items (e.g, an array).

The *doMC* package acts as an interface between *multicore* functionalities and the *foreach* package, which permits to execute looping operations on multiple processors.

R/parallel enables automatic parallelization of loops without data dependencies by exposing a single function: `runParallel()`. The implementation is based on C++, and combines low-level system calls to manage processes, threads and inter-process communications. The user defines which variable within the enclosed loop will store the calculation results after each iteration, and how these variables have to be operated and reduced.

It is worth to mention that an interface to Intel TBB for R also exists, that pretty much resembles our approach and permits to use TBB's `parallel_for` pattern to convert the work of a standard serial for loop into a parallel one, and the `parallel_reduce` construct can be used for accumulating aggregates or other values. This solution enforces a master/slave behaviour between R and C++, so that data-parallel computations can be offloaded to C++. We will shortly see that our approach pretty much resembles this latter one.

Memory management

The notoriously "poor" memory management mechanism in R is actually a combination of multiple factors, that also include the way operating systems allocate memory. Since our development relies on Linux OS, a discussion about these factors will shed some light over this problem.

R uses a *lazy* memory reclaim policy, in a sense that it will not reclaim memory until it is actually needed. Hence, R might be holding on to memory because the OS hasn't yet asked for it back, or it does not need more space yet.

In order to decide when to release memory, R uses a garbage collector (GC) that automatically releases memory to the OS when an object is no longer used. It does so by tracking how many references point to each object, and when there are no references pointing to an object it deletes that object and returns memory to the OS. This means that when we have one or more copies of a big object, explicitly removing the original object does not correspond to free memory space: until references to that object exists, the memory wont be released. Even a direct call to the GC does not force R to release memory, rather it acts as a "request", but R is free to ignore [7].

Furthermore, R has limited control over memory management mechanism: it simply uses `malloc/free` functions plus a garbage collector. One attempt to force memory to be released to the OS is the use of the `malloc_trim` function, that explicitly forces memory release, provided that a sufficiently large chunk is ready to be released. We managed to limit the drawbacks related to these weaknesses by avoiding unnecessary copies of objects and promptly freeing their memory, as soon as they are no longer needed. In this way we controlled memory leaks that cause memory fragmentation to explode.

1.2 Hi-C data analysis step-by-step

The Hi-C data analysis conducted with NuChart-II walks through five main steps:

1) data retrieval and parsing;
2) neighbourhood graph construction;
3) weighing of the edges as a result of data normalisation;
4) statistical analysis;
5) output and visualisation.

NuChart-II parses a number of options from Command Line Interface (CLI) to set up and characterise each execution. Once started, the application walks through all the steps outlined above in a "monolithic" fashion, and yields its results as a summary of the whole process: the final output is available in terms of a neighbourhood graph drawn using some plotting engine, together with formatted text files (such as `csv` files) that contain whole information necessary to examine the represented data. This include the actual sequences "contained" in edges, edges probability resulting from data normalisation, network analysis metrics and various statistical annotations.

Genomic data analysis, just like many other scientific fields, does not work as one monolithic process: different stages of data analysis are just fundamentally different, and have different parallelism patterns, memory access and data access requirements. Also, it often makes sense to run the same stage of an analysis in a number of different ways to demo nstrate the robustness of novel results or to tackle different sorts of data, for example one in which a reference genome is available, compared to one where it is not.

If we consider the possibility to map additional features on a graph — such as genes expression, CTCF binding sites or methylation profiles — we would choose

a dataset from which to gather the required information and re-execute the application from the beginning, until we get our output with mapped omics data. This means that no intermediate inspection is allowed, nor we could choose some quick statistics to satisfy whatever curiosity or to banish some doubts. Despite its undeniable efficiency, this lack of modularity highlights a clear limitation in usability of the C++ implementation.

These factors led us to re-consider R as a "hosting" environment for a scalable and usable tool for Hi-C data analysis. From the early R prototype — developed within the R environment — we learned that high-performance and good memory hierarchy exploitation is hard to achieve within the R environment, due to specificities of the environment itself, and requires a substantial programming effort. Nonetheless, research during the last decade has widely explored the use of parallel computing techniques with R.

2 Nucha*R*t

We aim at building a tool for Hi-C data analysis that is both efficient — in terms of speed and memory resources exploitation — and usable. We decided not to use off-the-shelf libraries for parallel computing, because of the well known R's limits in memory management: our search for long-range chromatin contacts over genome-wide paired-ends reads results in a memory-bound algorithm, thus parallel memory-intensive tasks should be kept on C++ side where we can obtain a finer memory control, while we rely on R for setting up a usable working environment. Also, we already had a fully tested C++ solution to our problem, that led us to consider Rcpp [4]: it facilitates data interchange from C++ to R and vice-versa. C++ objects holding the output of a computation are made available within the R environment, ready to be used as source for advanced statistical analysis, by mean of a *wrapping* mechanism based on the templated functions `Rcpp::as<>()` and `Rcpp::wrap()`. These functions convert C++ object classes into a *S expression pointers* (called `SEXP`), that can be handled on the R side to construct `Lists` or `DataFrames`[4], which are essential object types in R and are used by almost all modelling functions.

In this respect, our application clearly exhibits a *master/slave* behaviour: on the R side we set up the "background" for the computation, and then we offload computationally intensive tasks to C++ (see Figure 1). Once it terminates, the needed information is moved back to the R side and is ready to be processed, drawing from the huge R's library basket.

2.1 Nucha*R*t and Rcpp

In our context, we have dealt with four C++ objects that abstract the leading actors of our software: `SamData`, `Gene`, `Fragment` and `Edge`. These objects contain

[4] We actually use `data.table`s as basic data structures for our datasets: `data.table` is an enhanced version of `data.frame` that allows to easily optimise operations for speed and memory usage

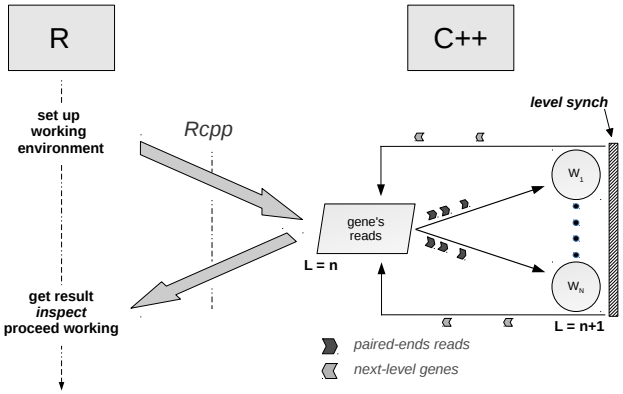


Fig. 1: Master/Slave behaviour between R and C++, on the graph construction phase

much of the information that is needed to build a topographical map of the DNA from Hi-C data. In order to exchange a `SamData` object between C++ and R we have specialised the templated functions above: a `std::vector<SamData>` is thus treated by R as a list of `Lists`, while a list of `Lists` in R (or a `DataFrame`) is managed in C++ by casting the `SEXP` object to a `Rcpp::List` (or a `Rcpp::DataFrame`) object, and by subsequently filling each field of the `SamData` class with the value contained in the respective field of the `List`.

Recalling section 1.2, NuChart-II can be described as a 5 stages pipeline: from data retrieval to output and visualisation, these phases can now be broke up and used as loose modules. Phase *1)* is responsible for data collection and early data processing: datasets are provided as static `csv`-like files, but can also be downloaded from on-line repositories. The information contained therein is parsed and processed, in order to build the data collections needed to perform the computations: unneeded fields are dropped and elements are ordered in a consistent way, while a unique identifier for each element of a collection is generated, when needed. Problems may arise if these operations are performed on the R side, as it may lead to memory overflows with big size files ($> 2GB$, as it is the case with SAM files) due to the way R objects are constructed and stored in memory. For this reason dataset whose size exceeds $2GB$ is parsed on C++ side (as it is the case for `SAM` files). No matter where these operations are executed, objects can be moved from C++ side to R side and vice-versa, as explained above.

Phases *2)* and *3)* constitute by far the most onerous parts of the application, in terms of execution time and resource consume. Both of them are suitable for being revisited in the context of loop parallelism, since their kernels can be run concurrently on multiple processors with no data dependencies involved. These

Algorithm 1.1 Example of `as` and `wrap` usage

```
template<> SEXP wrap(const SamData &s) {
    List ret = List::create( Named("Id")    = s.getId(),
                             Named("Chr1")  = s.getChr1(),
                             Named("Start1") = s.getStart1(),
                             Named("Chr2")  = s.getChr2(),
                             Named("Start2") = s.getStart2(),
                             Named("Seq")   = s.getSeq()
                           );
    return wrap(ret);
}

template<> SamData as( SEXP s ) {
    List samL = as<List>(s);
    SamData sam;

    sam.setId     (  as<long>( samL["Id"])    );
    sam.setChr1   (  as<string>(samL["Chr1"])  );
    sam.setStart1 (  as<long>( samL["Start1"]) );
    sam.setChr2   (  as<string>(samL["Chr2"]) );
    sam.setStart2 (  as<long>( samL["Start2"]) );
    sam.setSeq    (  as<string>(samL["Seq"])  );

    return sam;
}
```

phases have been thoroughly explained in our previous works [3,5,6], and we refer to those writings for a thorough explanation. Not much changes when we offload the a computation from R to C++: the very same logic is used and the `ParallelFor` skeleton permits to speed up both phases in a seamless way. Data transfer overhead is negligible, compared to the computationally intensive task that takes place.

Phase *4)* encompasses essential features that the package ought to provide, in order to fulfil the requirements of a useful tool for genomic data interpretation. With a graph-based representation we can apply network analysis over the resulting graph: topological measures capture graph's structure for nodes and edges and highlight the "importance" of the actors. For instance, centrality metrics describe the interactions that (may) occur among local entities. Ranking of nodes by topological features (such as degree distribution) can help to prioritize targets of further studies or lead to a more local, in-deep analysis of specific chromosome locations. Here studies of functional similarity can suggest new testable hypotheses [8].

Finally, visualisation is crucial for a tool that aims at facilitating a better interpretation of genomic data. NuChart-II supplies both tabular output and graphical visualization. Concerning the latter, common plotting engines perform nicely with small-to-medium sized graphs, but cannot provide useful representations of huge graphs.

A possible approach could be to decouple visualisation from Nucha*R*t, and make use of external applications purposely designed for interactive visualisation

of networks. One such application is *Gephi*[5], that permits to interact with the graph by manipulating structures, shapes and colors to reveal hidden properties. Nucha*R*t can output a resulting graph in `GraphML` format, which permit to get the most out of Gephi. In this way the user can easily browse the results of Hi-C data analysis through Gephi interface.

3 Discussion

The novel package benefits of the combined use of parallel programming techniques provided by the C++ engine, and the flexibility of the R environment, maintaining the same performance and scalability achieved in NuChart-II. Moreover, within the R environment the five steps listed in section 1.2 become loose but totally compatible modules, and could be either executed in order or as services that permit to accomplish a specific task.

Results of each module are made globally available in form of `DataFrame`s, and can be easily queried and inspected, exported, or saved and re-used with other, different data analysis tools. The graph can be plotted and the results can be visualised and browsed. Eventually, one can draw from the huge R's libraries basket the one that suits her need, and conduct advanced analysis over the resulting data. For instance, we also tested the *ERGM* package that permit to understand the processes of network structure emergence and tie formation: the Exponential-family Random Graph Models package provides an integrated set of tools for creating an estimator of the network through a stochastic modelling approach.

3.1 Experiments

The study of the interactions of the actor genes with the environment is of critical importance for understanding the entire system. By using the modelling functions of the package we can statistically characterize the distribution of the edges in relation to the characteristics of the nodes that represent mapped multi-omics features. We performed the analysis of the clusters of genes Human Leukocyte Antigen (HLA, Figure 3) and Kruppel-Associated Box (KRAB, Figure 2) in the context of four Dixon experiments (SRA:SRR400260, SRA:SRR400261, SRA:SRR400266, SRA:SRR400267) [2], to verify the correlation of the edges distribution in relation to some genomic features (hypersensitive sites, CTCF binding sites, isochores, RSSs).

The first analysed locus is located in cytoband chr19.q13.12 and concerns the clusters of Kruppel-type zinc finger genes, related to the KRAB, which are peculiar for their tandem organization. Zinc finger proteins are a family of transcription factors that regulate the gene expression, and most of these proteins are members of the KZNF family. There are 7 human-specific novel KZNFs and 10 KZNFs that have undergone pseudo-gene transformation specifically in the

[5] https://gephi.org/

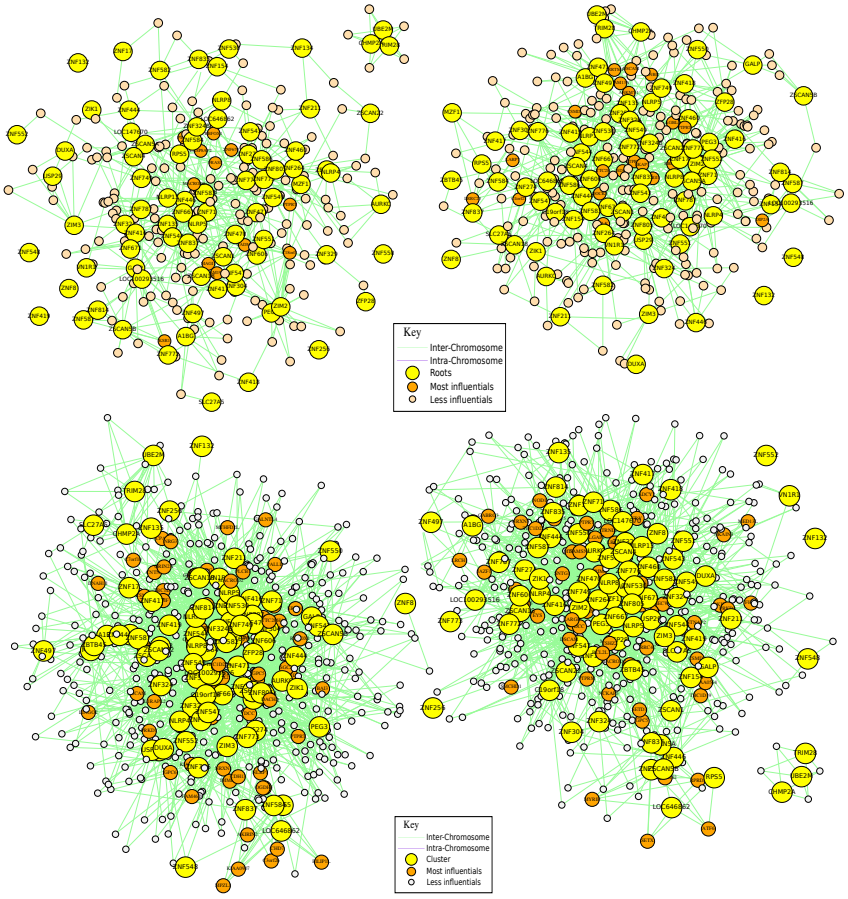


Fig. 2: Neighbourhood graphs of the KRAB cluster of genes in four different runs from the Hi-C experiments of Dixon et al.

human lineage. 30 additional KZNFs have experienced human-specific sequence changes that are presumed to be of functional significance. Members of the KZNF family are often in regions of segmental duplications, and multiple KZNFs have undergone human-specific duplications and inversions. In Figure 2, top panel drawings concern sequencing runs from hESC (SRR400260, SRR400261); bottom panel drawings in the same Figure concern sequencing runs from IMR90 (SRR400266, SRR400267). Seed genes are the genes given as input to the algorithm, while output genes are differentially represented according to their importance (in terms of node degree).

The second analysed gene cluster concerns the human leukocyte antigen (HLA) system, which is the name of the locus containing the genes that encode for major histocompatibility complex (MHC) in humans. It belongs to a super-locus that contains a large number of genes related to the immune system

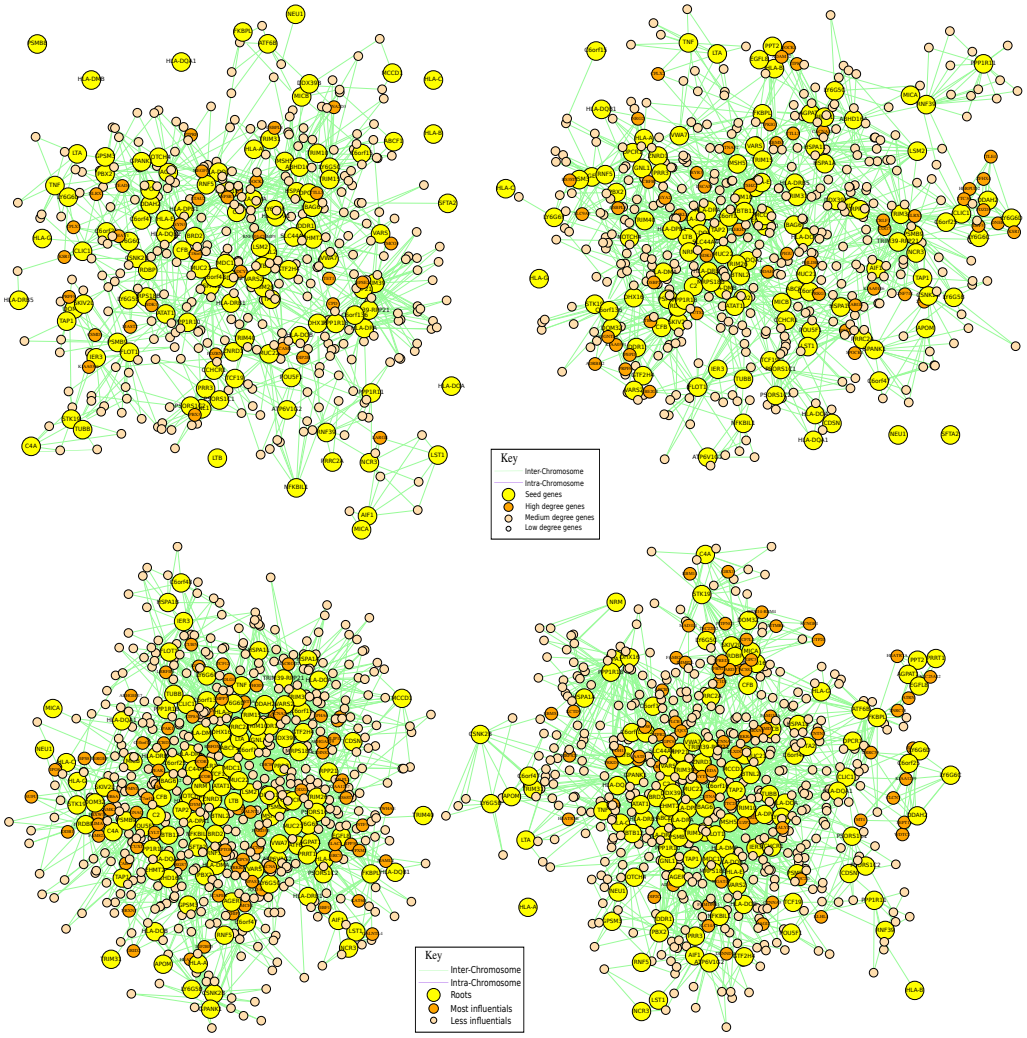


Fig. 3: Neighbourhood graphs of the HLA cluster of genes in four different runs from the Hi-C experiments of Dixon et al.

function in humans. The HLA group of genes resides on cytoband chr6.p31.21 and encodes for cell-surface antigen-presenting proteins, which have many different functions. The HLA genes are the human version of the MHC genes that are found in most vertebrates (and thus are the most studied of the MHC genes). The major HLA antigens are essential elements for the immune function. In Figure 3, top panel drawings concern sequencing runs from hESC (SRR400260, SRR400261); bottom panel drawings in the same Figure concern sequencing runs from IMR90 (SRR400266, SRR400267). Seed genes are the genes given as input

to the algorithm, while output genes are differentially represented according to their importance (in terms of node degree).

The correlation between cryptic RSS sites and edges is more pronounced in the HLA cluster, in comparison to the KRAB cluster, probably due to a more consistent presence of this kind of sequences in genes related to the immune system. The correlation between hypersensitive sites (super sensitivity to cleavage by DNase) and edges, although positive, is poor, probably because the accessibility of these regions are impaired by a large number of long-range interactions. The correlation between the presence of CTCF binding sites and edges was clearly predictable, because linking Gene-Regulatory elements maintain different regions of the genome close to each other. On the other hand, regions with isochores seem less involved in long-range interactions, which can be quite surprising considering that these portions of the genome are considered gene-rich.

Statistical results are reported in table 1. The network estimators are all computed using 100 iterations of stochastic modelling. A high correlation between the presence of specific genomic features and the probability of existence of an edge persists. DNase sensitivity sites are weakly correlated with the presence of an edge, while isochores are strongly anti-correlated with the presence of an edge.

Performance comparisons between the original R prototype and the actual implementation have not been conducted, because there would be no room for such debate, since the original tool often halted its execution, due to its strong limitations in memory management. For what it concerns the comparison between the C++ application and the combined R with C++ package, they report substantially similar behaviours: the graph construction execution is strongly affected by datasets size and resolution, that determine the "search space" for the BFS-like graph construction and the overall memory load. Reducing the working set ameliorates execution times and overall scalability with NuChart-II, and clearly helps in obtaining good performance when offloading the graph construction from R to C++ [3].

Figure 4 compares execution time (left) and speedup (right) in the two approaches: Figures 4a and 4b show the performance for constructing a graph at level 1 starting from the KRAB cluster of genes using Dixon's SRR400266 experiment as Hi-C dataset. Despite similar timings and scalability, Nucha*R*t has slightly worse performance and shows a higher execution time. Figures 4c and 4d show a comparison of the performance during normalisation phase with NuChart-II and Nucha*R*t: again both implementations yield similar results, both approaching a quasi-linear scalability, even though Nucha*R*t's execution time is slightly higher with respect to NuChart-II's. This is likely due to the worsening of memory access time when offloading computation to C++: while the multi-threaded C++ application is running, the R environment is kept alive. R stores additional information, beside the data itself, for each object created: when this small overhead is combined to the lazy memory reclaim policy adopted by R's garbage collector, and to the massive size of the dataset used for neighbourhood

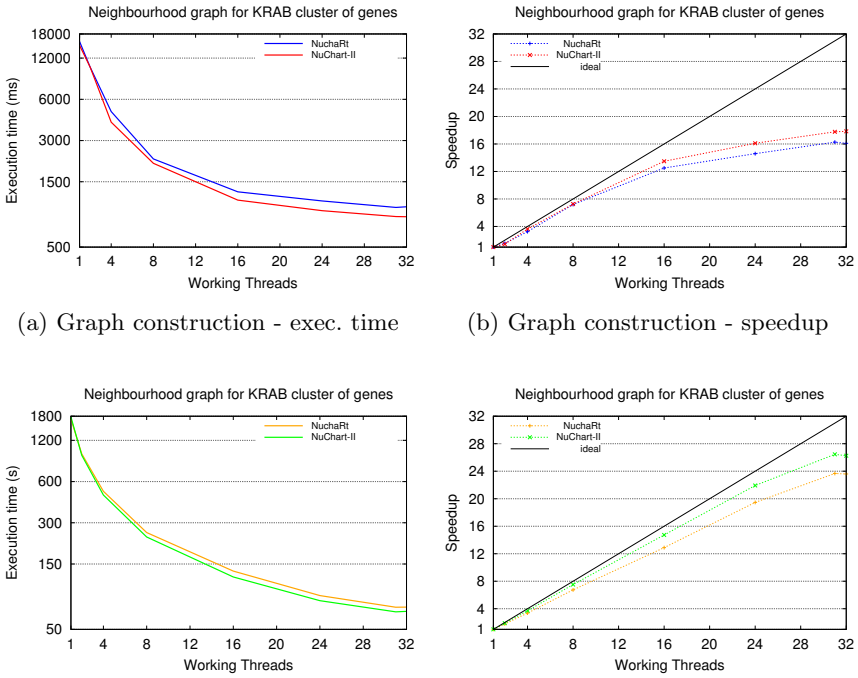
Table 1: Mapping CTCF binding sites, isochores, cryptic RSSs, and DNase sites on the graphs affects the edge distribution of the KRAB cluster of genes and of the HLA cluster of genes, using the ERGM package

	KRAB		HLA	
	Estimate	**Std. Error**	**Estimate**	**Std. Error**
SRA:SRR400260				
edges + nodecov("dnase")	0.2867	0.08451	0.1711	0.07961
edges + nodecov("ctcf")	0.6531	0.01157	0.5545	0.01253
edges + nodecov("rss")	0.5804	0.06176	0.6304	0.08196
edges + nodecov("iso")	-1.047	0.09269	-0.9406	0.09156
SRA:SRR400261				
edges + nodecov("dnase")	0.2042	0.07932	0.1706	0.07822
edges + nodecov("ctcf")	0.6629	0.04158	0.5687	0.02005
edges + nodecov("rss")	0.5378	0.03566	0.6319	0.03776
edges + nodecov("iso")	-1.015	0.09566	-0.93035	0.08969
SRA:SRR400266				
edges + nodecov("dnase")	0.2042	0.07932	0.1706	0.07822
edges + nodecov("ctcf")	0.6629	0.04158	0.5687	0.02005
edges + nodecov("rss")	0.5378	0.03566	0.6319	0.03776
edges + nodecov("iso")	-1.015	0.09566	-0.93035	0.08969
SRA:SRR400267				
edges + nodecov("dnase")	0.2042	0.07932	0.1706	0.07822
edges + nodecov("ctcf")	0.6629	0.04158	0.5687	0.02005
edges + nodecov("rss")	0.5378	0.03566	0.6319	0.03776
edges + nodecov("iso")	-1.015	0.09566	-0.93035	0.08969

graph construction, resident memory consumption remains high at run-time, thus affecting memory access times and overall performance.

Our experiments where conducted on a NUMA system, equipped with 4 eight-cores E7-4820 Nehalem running at 2.0GHz, with 18MB L3 cache and 64 GB of main memory. The Nehalem processor has Hyper-Threading capability with 2 contexts per core, but we decided not to use it and stick to the number of physical cores: the heavy memory usage would dramatically damage performance, and likely increase chances of false-sharing among threads in the same context that share L2 cache. With this configuration the cache-coherence mechanism plays an important role in this performance degradation, where cache misses are likely frequent and cache lines updates occur frequently.

Performance differences seem to flatten when the same applications are executed on a different machine: we also conducted experiments on a workstation equipped with a single eight-cores Intel Xeon CPU E5-2650 running at 2.60GHz. This machine features 20MB of L3 cache with 64 GB of main memory. The SandyBridge processor also has Hyper-Threading capability allowing 2 contexts per core. Here as well we decided to not run more than 8 threads, so that the



(a) Graph construction - exec. time

(b) Graph construction - speedup

(c) Normalisation - exec. time

(d) Normalisation - speedup

Fig. 4: Comparison between NuChart-II and Nucha*R*t during the graph construction and normalisation

second context is not used an only physical cores are employed during computation. In this case the gap between the two solutions is reduced, though the total execution time is higher due to the limited degree of parallelism that can be achieved because of the reduced number of available cores.

4 Concluding remarks

Embedding NuChart-II in R creates an application that can be used either to conduct a step-by-step analysis of genomic data, or as a high-performance workflow that takes heterogeneous datasets in input, processes data and produces a graph-based representation of the chromosomal information provided, supported by a rich set of default descriptive statistics derived from the topology of the graph. The graph-based approach fosters a tight coupling of topological observations to biological knowledge, which is likely to bring remarkable biological insights to the whole research community.

From a computational point of view, the ever-increasing amount of information generated by novel Bioinformatics techniques require proper solutions that permit the full exploitation of the computing power offered by modern

computing systems, together with advanced tools for an efficient analysis and interpretation of genomic data. These tasks require high skills, but we believe that Nucha*R*t can be a valuable mean to support researchers in pursuing these objectives.

Despite the results achieved in terms of performance and usability, some problems remain partially unsolved, and are open to further investigations. Among them, our main concern is the visualisation of multi-omic graphs, which we believe is an essential feature for a usable tool aimed at facilitating genomic data analysis and interpretation, and that remains an open problem.

We are currently working on making the package compliant to Bioconductor and CRAN requirements, so that it can be easily downloaded and used by the research community. At the time of writing it is available through our research group's repository, at `http://alpha.di.unito.it:8080/tordini/nuchaRt`.

Acknowledgement This work has been partially supported by the EC-FP7 STREP project "REPARA" (no. 609666), the Italian Ministry of Education and Research Flagship (PB05) "InterOmics", and the EC-FP7 innovation project "MIMOMICS".

References

1. Dekker, J., Rippe, K., Dekker, M., Kleckner, N.: Capturing Chromosome Conformation. Science 295(5558), 1306–1311 (Feb 2002)
2. Dixon, J., Selvaraj, S., Yue, F., Kim, A., Li, Y., Shen, Y., Hu, M., Liu, J., Ren, B.: Topological domains in mammalian genomes identified by analysis of chromatin interactions. Nature 485(5), 376–80 (2012)
3. Drocco, M., Misale, C., Peretti Pezzi, G., Tordini, F., Aldinucci, M.: Memory-optimised parallel processing of Hi-C data. In: Proc. of Intl. Euromicro PDP 2015: Parallel Distributed and network-based Processing. pp. 1–8. IEEE (Mar 2015), `http://calvados.di.unipi.it/storage/paper_files/2015_pdp_memopt.pdf`
4. Eddelbuettel, D.: Seamless R and C++ Integration with Rcpp. Springer, New York (2013), iSBN 978-1-4614-6867-7
5. Merelli, I., Liò, P., Milanesi, L.: Nuchart: An r package to study gene spatial neighbourhoods with multi-omics annotations. PLoS ONE 8(9), e75146 (09 2013)
6. Tordini, F., Drocco, M., Misale, C., Milanesi, L., Liò, P., Merelli, I., Aldinucci, M.: Parallel exploration of the nuclear chromosome conformation with NuChart-II. In: Proc. of Intl. Euromicro PDP 2015: Parallel Distributed and network-based Processing. IEEE (Mar 2015), `http://calvados.di.unipi.it/storage/paper_files/2015_pdp_nuchartff.pdf`
7. Wickham, H.: Advanced R. Chapman and Hall/CRC, 1 edn. (Oct 2014)
8. Winterbach, W., Mieghem, P.V., Reinders, M.J.T., Wang, H., de Ridder, D.: Topology of molecular interaction networks. BMC Systems Biology 7, 90 (2013)