StoHi-C: Using t-Distributed Stochastic Neighbor Embedding (t-SNE) to predict 3D genome structure from Hi-C Data

Kimberly MacKay∗,1 and Anthony Kusalik∗
∗Department of Computer Science, University of Saskatchewan, Saskatoon, SK, Canada, S7N 5C9

ABSTRACT In order to comprehensively understand the structure-function relationship of the genome, 3D genome structures must first be predicted from biological data (like Hi-C) using computational tools. Many of these existing tools rely partially or completely on multi-dimensional scaling (MDS) to embed predicted structures in 3D space. MDS is known to have inherent problems when applied to high-dimensional datasets like Hi-C. Alternatively, t-Distributed Stochastic Neighbor Embedding (t-SNE) is able to overcome these problems but has not been applied to predict 3D genome structures. In this manuscript, we present a new workflow called StoHi-C (pronounced "stoic") that uses t-SNE to predict 3D genome structure from Hi-C data. StoHi-C was used to predict 3D genome structures for multiple, independent existing fission yeast Hi-C datasets. Overall, StoHi-C was able to generate 3D genome structures that more clearly exhibit the established principles of fission yeast 3D genomic organization.

KEYWORDS 3D Genome Reconstruction Problem; 3D Genomics; 3D Genome Structure; 3D Genome Organization; t-Distributed Stochastic Neighbor Embedding; Hi-C; Fission Yeast

INTRODUCTION Understanding the structure-function relationship of various biomolecules has been the foundation of molecular biology research for many years. Recently, the development of Hi-C (and related methods) has resulted in the generation of unprecedented sequence-level investigations into the structure-function relationship of the genome (Lieberman-Aiden et al. 2009; Belton et al. 2012; Belaghzala et al. 2017). Hi-C is able to detect regions of the genome that are "interacting" (i.e. in close 3D spatial proximity). Typically, this is done by mapping Hi-C sequence reads to a reference genome (Lajoie et al. 2015; Wingett et al. 2015; MacKay et al. 2018; MacKay and Kusalik 2019). This results in the generation of a whole-genome contact map which is a N × N matrix where N is the number of "bins" which represent linear regions of genomic DNA. Each cell within a whole-genome contact map records a count of how many times two genomic regions were found in close proximity within a population of cells (Lajoie et al. 2015; MacKay et al. 2018; MacKay and Kusalik 2019). This is more commonly referred to as an interaction count. Interaction counts are often normalized using methods like iterative correction and eigenvector (ICE) decomposition (Imakaev et al. 2012; Varoquaux and Servant 2019) to reduce inherent biases within Hi-C datasets (Yang and Jiang 2014; Li et al. 2015; Servant et al. 2015; Stansfield et al. 2018; Lyu et al. 2019; Spill et al. 2019). This normalization process results in fractional interaction counts also known as interaction frequencies.

Normalized whole-genome contact maps can be used to infer 3D genomic structure(s). This is known as the 3D genome reconstruction problem (3D-GRP) (Segal and Bengtsson 2015; MacKay and Kusalik 2019) or the 3D chromatin structure modelling problem (Zhang et al. 2013). For the purpose of this manuscript, we will be using the term 3D-GRP. A formal representation of the 3D-GRP is provided by MacKay and Kusalik (2019). Briefly, normalized interaction frequencies are converted into a set of pairwise distances (based on the inverse of the interaction frequency). This calculation uses the assumption that a pair of genomic regions with a small interaction frequency will be further away in 3D space than a pair
of genomic regions with a higher interaction frequency (Duan et al. 2010; Fraser et al. 2010; Rousseau et al. 2011; Bait and Marti-Renom 2011, 2012; Hu et al. 2013; Ay et al. 2014; Lesne et al. 2014; Varoquaux et al. 2014; Sekelija et al. 2016; MacKay and Kusalik 2019). Each genomic bin’s \((x, y, z)\) coordinates are then calculated using various optimization techniques (MacKay and Kusalik 2019).

Many of the existing tools for solving the 3D-GRP rely on multi-dimensional scaling (MDS) either completely or partially to predict and embed genomic structures in 3D space. MDS is known to have inherent problems when calculating embeddings from population-based, sparse, high-dimensional datasets (which are characteristics of Hi-C datasets) (Adhikari et al. 2016; Rieber and Mahony 2017). Alternatively, t-Stochastic Neighbourhood Embedding (t-SNE) has resulted in more accurate embeddings for datasets with these characteristics (van der Maaten and Hinton 2008; van der Maaten 2009; van der Maaten and Hinton 2012; van der Maaten 2014). Recently, Zhu et al. (2018) were able to predict 3D structures of individual chromosomes using a manifold-learning approach (similar to t-SNE) combined with multi-conformation optimization. Their tool was shown to outperform many of the existing MDS-based methods but could not be applied to the entire 3D-GRP due to the underlying time complexity of multi-conformation optimization (Zhu et al. 2018). Based on the results of this regional prediction tool, t-SNE should result in more accurate solutions to the 3D-GRP when compared to existing MDS methods. To test this hypothesis, we developed a new workflow called StoHi-C (pronounced “stoic”) that uses t-SNE to predict 3D genome structure from Hi-C data. StoHi-C and MDS were used to predict 3D genome structure for four existing fission yeast datasets (wild-type, G1-arrested, rad21 deletion and clr4D deletion). Overall, StoHi-C was able to more clearly recapitulate well-documented features of fission yeast chromosomal organization (such as the RabI structure) when compared to the MDS method.

METHODS

StoHi-C is a two step workflow that involves (1) 3D embedding and (2) visualization. A more detailed description of each step is provided in the subsections below. Each step can be run independently or users can invoke an automated shell script \(^1\) that runs each step in succession. Complete documentation describing expected inputs, outputs and software requirements can be found on the project homepage \(^2\). In the subsequent sections, the StoHi-C workflow is described in general, but also provides details regarding the specific illustrative examples presented in this paper.

Step 1: 3D Embedding

The 3D coordinates for each genomic bin are calculated using t-SNE (van der Maaten and Hinton 2008; van der Maaten 2009; van der Maaten and Hinton 2012; van der Maaten 2014). A python script \(^3\) was developed that accepts a normalized whole-genome contact map as input and outputs the \((x, y, z)\) coordinates for each genomic bin. An example of the required input and expected output can be found on the project homepage. This script uses the TSNE method from the sklearn.manifold library to embed genomic bins in 3D space. The exact parameter values that were used for the fission yeast datasets as well as a brief description of their function follow: `n_components = 3` (embedding dimensionality), `perplexity=5.0` (number of nearest neighbours), `early_exaggeration=3.0` (controls the tightness of clusters), `n_iter=5000` (maximum number of iterations), `method='exact'` (do not use the Barnes-Hut approximation), `init='pca'` (use a principle component analysis to initialize the embedding). These values were selected based on the suggestions provided on t-SNE’s homepage \(^4\).

Step 2: Visualization

Once the \((x, y, z)\) coordinates are generated a multitude of different tools can be used for visualization. Three options are discussed below but any graphing or network visualization tool that accepts 3D coordinates (where \(x, y, z\) values are space-delimited with each point on a separate row) could be used.

1. plot.ly: A python script plotly_viz.py \(^5\) was developed that accepts the \((x, y, z)\) coordinates generated in Step 1 and produces a static PNG image and an interactive 3D graph (HTML) using the plot.ly library. (Plotly Technologies Inc 2015). The interactive graph can be opened in any web browser. This option was used to generate the figures for the illustrative examples in this manuscript.

2. matplotlib: A python script matplotlib_viz.py \(^6\) was developed that accepts the \((x, y, z)\) coordinates generated in Step 1 and produces a static PNG image of the corresponding 3D graph as well as a simple MP4 animation that rotates around the y-axis. This script uses the py.plot and animation modules from the matplotlib library (Hunter 2007) as well as the mpl_toolkits.mplot3d toolkit \(^7\).

3. Chart Studio: Alternatively, plot.ly has a web-based, interactive version available online called Chart Studio \(^8\). The 3D coordinates can be directly uploaded to the website to generate an interactive graph. Chart Studio has provided a detailed tutorial on generating this type of visualization \(^9\). Customized styles such as colours, labels, size, transparency, etc. for nodes and/or edges can then be set directly by the user through the graphical user interface. Additional node and/or edge attributes can be added to the 3D graph to incorporate complementary biological datasets (if available) with the visualization.

Options 1 and 2 have been automated for the visualization of fission yeast datasets with 10kb resolution. Applying them to datasets from other organisms or datasets with different resolutions would require slight adjustments to the scripts. Documentation of how to make these changes is provided on the project homepage \(^10\). Option 3 can not be automated since it is a graphical user interface.

Comparison with MDS

In order to compare the results of StoHi-C with MDS, the generation of \((x, y, z)\) coordinates in step 1 was also done with metric-MDS. The use of metric-MDS for 3D genome prediction has been widely used since 2010 (Duan et al. 2010; Tanizawa et al. 2010). Similarly to Step 1 of the StoHi-C workflow, a python script was developed \(^11\) that accepts a normalized whole-genome contact map as input and outputs the \((x, y, z)\) coordinates for each genomic bin. This script uses the MDS method from the sklearn.manifold library (Pedregosa et al. 2011) to embed genomic bins in 3D space. The exact parameter values that were used for the fission yeast datasets as well as a brief description of their function follow: `n_components = 3` (embedding dimensionality), `metric=True` (use metric MDS), `max_iter=5000` (maximum number of iterations), `dissimilarity='precomputed'` (use a custom dissimilarity matrix). To be consistent with the StoHi-C workflow, the plot.ly script used for Step 2 (described above) was
used to visualize the results for the illustrative examples in this manuscript.

Data Availability
The datasets supporting the conclusions of this article were originally generated by Mizuguchi et al. (2014) and are available in the Gene Expression Omnibus database (accession number: GSE56849). The specific sample numbers are 999a wild-type (GSM1379427), G1-arrested (GSM1379429), rad21-K1 mutation (GSM1379430) and clr4 deletion (GSM1379431).

Web Resources
StoHi-C is freely available at https://github.com/kimmackay/StoHi-C and is licensed under the Creative Commons Attribution-NonCommercial-ShareAlike 3.0. It requires Python3 and local copies the following libraries: numpy, sklearn and plot.ly. These libraries are open access and can be downloaded through a package manager like pip or conda. Archived versions of the scripts used to generate the results in this manuscript are available as supplemental data.

RESULTS & DISCUSSION
The StoHi-C workflow and MDS method described above were used to generate 3D genome predictions for four existing Hi-C fission yeast datasets (999a wild type, G1-arrested, rad21-K1 mutation, and clr4 deletion) (Mizuguchi et al. 2014). Depending on the method, either t-SNE or MDS was used to generate (x, y, z) coordinates. These results were then visualized with plotly which generates both static images and interactive graphs. Images representing the genomic predictions for each dataset with the StoHi-C workflow (Panels A,C,E,G) and MDS method (Panels B,D,F,H) are presented in Figure 1. Interactive versions of each plotly graph can be found on the project homepage. Additionally, the project homepage contains the resultant images and animations generated with the matplotlib visualization for the 999a wild-type dataset.

Based on the results presented in Figure 1, the 3D genomic predictions generated with StoHi-C more clearly represent known features of fission yeast chromosomal organization when compared to the MDS method. The StoHi-C predictions (Figure 1A,C,E,G) all clearly depict universal hallmarks of genome organization (e.g. chromosome territories (Cremer and Cremer 2010)) as well as fission yeast specific features (e.g. Rabl configuration (Mizuguchi et al. 2015; Fernández-Alvarez and Cooper 2017)). Meanwhile, the MDS predictions all resulted in a hairball-like configuration with no apparent biological significance (Figure 1B,D,F,H). This is likely due to a fundamental difference in the algorithms underlying t-SNE and MDS. One of the goals of t-SNE is to preserve the local structure of high-dimensional datasets by placing similar features close together in the final embedding (van der Maaten and Hinton 2008; van der Maaten 2009; van der Maaten and Hinton 2013). Meanwhile, StoHi-C was able to produce 3D genomic predictions that are consistent with the large body of work depicting fission yeast genomic organization including the Rabl configuration. This is the first time the Rabl configuration has been successfully predicted from fission yeast Hi-C data. This is surprising when considering the relative simplicity of the fission yeast genome, but more understandable due to existing tools heavy reliance on MDS. It should be noted that polymer modelling of the same datasets was not successful (Mizuguchi et al. 2014).

While StoHi-C appears to be working well with data from the haploid organism fission yeast, additional steps may be required to apply it to organisms with higher ploidy (diploid, hexaploid, etc.) if the data is not pre-phased. This is because StoHi-C will have to determine which chromosome copy (or copies) contribute to the detected interactions (the ploidy problem). For now, users should preprocess polyplody Hi-C data with existing phasing tools (see review by Browning and Browning (2011)) prior to using StoHi-C. To solve this problem more permanently, future work will focus on extending StoHi-C to include a step that performs phasing. This is something we are actively working toward in the hopes of applying StoHi-C to polyplody organisms. Once this has been completed, it will be deployed as a new version on the project homepage.

In this manuscript, we present a new workflow called StoHi-C required to embed and visualize each dataset using both the StoHi-C workflow and MDS method. These timings do not represent a comprehensive complexity analysis and instead are presented to provide context as to whether or not these methods are practical for Hi-C-sized datasets. Interestingly, the MDS embedding is much faster than the t-SNE embedding with average elapsed times of 0.53 seconds and 11.0 seconds, respectively. This could be due to efficiencies in the implementations of the two algorithms.

Table 1 Elapsed Runtimes for 3D genome prediction with the StoHi-C workflow and MDS method. Elapsed runtimes are shown in seconds for the embedding (step 1), visualization (step 2) and complete workflow (total).

| Dataset | StoHi-C Elapsed Time | MDS Elapsed Time |
|---------|----------------------|-----------------|
|         | Step 1 | Step 2 | Total | Step 1 | Step 2 | Total |
| 999a Wild-Type | 10.6 | 5.7 | 16.2 | 0.54 | 5.7 | 6.2 |
| G1-Arrested | 11.7 | 6.3 | 18.0 | 0.52 | 5.9 | 6.4 |
| rad21-K1 Mutation | 10.8 | 6.3 | 17.1 | 0.54 | 5.8 | 6.3 |
| clr4 Deletion | 10.7 | 5.8 | 16.5 | 0.51 | 5.6 | 6.1 |

To the best of our knowledge, none of the existing methods for predicting 3D genomic organization have been successfully applied to the datasets used in this paper. Previously, Tanizawa et al. (2010) applied MDS to chromosome conformation capture data from fission yeast but the results were not able to recapitulate the Rabl configuration because of the Rabl configuration. StoHi-C was able to produce 3D genomic predictions that are consistent with the large body of work depicting fission yeast genomic organization. StoHi-C is the first time the Rabl configuration has been successfully predicted from fission yeast Hi-C data. This is surprising when considering the relative simplicity of the fission yeast genome, but more understandable due to existing tools heavy reliance on MDS. It should be noted that polymer modelling of the same datasets was not successful (Mizuguchi et al. 2014).
Figure 1 Visualizations of 3D genome predictions for four fission yeast datasets using StoHi-C and MDS. Panels A,C,E and G show 3D genome predictions produced with the StoHi-C workflow, while Panels B,D,F, and H depict the 3D genome predictions generated with MDS. In all panels, chromosomes are indicated with the following colours: purple (chromosome 1), orange (chromosome 2), green (chromosome 3). Corresponding datasets are indicated in the black box directly above the panels (999a wild-type: Panels A and B, G1-arrested: Panels C and G, rad21-K1 mutation: Panels E and F, clr4 deletion: Panels G and H). In each panel, the X, Y and Z axes are indicated with a corresponding label.

(pronounced "stoic") that uses t-SNE to predict 3D genome structure from Hi-C data. Unlike MDS, t-SNE is well-suited for embedding population-based, sparse, high-dimensional data in 3D space. StoHi-C was used to predict 3D genome structures for four fission yeast Hi-C datasets. The results were compared to the 3D genomic structures predicted from the same datasets using a MDS approach. The 3D genomic predictions generated with StoHi-C more clearly represent known features of fission yeast chromosomal organization when compared to the MDS method. Additionally, this is the first time the Rab1 3D genomic organization was successfully predicted from fission yeast Hi-C data. Overall, StoHi-C was able to generate 3D genome structures that more clearly exhibit the established principles of fission yeast 3D genomic organization when compared to the MDS results.

ENDNOTES
1. https://github.com/kimmackay/StoHi-C/blob/master/stohic.sh
2. https://github.com/kimmackay/StoHi-C/
3. https://github.com/kimmackay/StoHi-C/blob/master/step1/tSNE/run_tSNE.py
4. https://lvdmaaten.github.io/tsne/
5. https://github.com/kimmackay/StoHi-C/blob/master/step2/plotly_viz.py
6. https://github.com/kimmackay/StoHi-C/blob/master/step2/matplotlib_viz.py
7. https://matplotlib.org/mpl_toolkits/mplot3d/index.html#mpl_toolkits.mplot3d
8. https://chart-studio.plot.ly/create/#/
9. https://plotly.io/make-a-3d-scatter-plot/
10. https://github.com/kimmackay/StoHi-C/issues
11. https://github.com/kimmackay/StoHi-C/blob/master/step2/MDS_results/matplotlib/999a_WT
12. https://www.ncbi.nlm.nih.gov/geo/query/acc.cgi?acc=GSE56849
13. https://github.com/kimmackay/StoHi-C/tree/master/interactive_visualizations/
14. https://github.com/kimmackay/StoHi-C/tree/master/step2/tSNE_results/matplotlib/999a_WT
15. https://github.com/kimmackay/StoHi-C/tree/master/step2/MDS_results/matplotlib/999a_WT

ACKNOWLEDGEMENTS
This work was supported by the Natural Sciences and Engineering Research Council of Canada [RGPIN 37207 to AK, Vanier Canada Graduate Scholarship to KM].

LITERATURE CITED
Adhikari, B., T. Trieu, and J. Cheng, 2016 Chromosome3D: reconstructing three-dimensional chromosomal structures from Hi-C interaction frequency data using distance geometry simulated annealing. BMC Genomics 17: 3210–3214.
Ay, F., E. M. Bunnik, N. Varoquaux, S. M. Bol, J. Prudhomme, et al., 2014 Three-dimensional modeling of the P. falciparum genome
during the erythrocytic cycle reveals a strong connection between genome architecture and gene expression. Genome Research 24: 974–988.

Baù, D. and M. A. Marti-Renom, 2011 Structure determination of genomic domains by satisfaction of spatial restraints. Chromosome Research 19: 25–35.

Baù, D. and M. A. Marti-Renom, 2012 Genome structure determination via 3C-based data integration by the Integrative Modeling Platform. Methods 58: 300–306.

Belaghzala, H., J. Dekker, and J. H. Gibcus, 2017 Hi-C 2.0: An optimized Hi-C procedure for high-resolution genome-wide mapping of chromosome conformation. Methods 123: 56–65.

Belton, J.-M., R. P. McCord, J. H. Gibcus, N. Naumova, Y. Zhan, et al., 2012 Hi–C: A comprehensive technique to capture the conformation of genomes. Methods 58: 268–276.

Browning, S. R. and B. L. Browning, 2011 Haplotype phasing: existing methods and new developments. Nature Reviews Genetics 12: 703–714.

Cremer, T. and M. Cremer, 2010 Chromosome territories. Cold Spring Harbor Perspectives in Biology 2: a003889.

Duan, Z., M. Andronescu, K. Schutz, S. McLlwan, Y. J. Kim, et al., 2010 A three-dimensional model of the yeast genome. Nature 465: 363–367.

Fernández-Álvarez, A. and J. P. Cooper, 2017 The functionally elu- sive rabi chromosome configuration directly regulates nuclear membrane remodeling at mitotic onset. Cell Cycle 16: 1392–1396.

Fraser, J., M. Rousseau, M. Blanchette, and J. Dostie, 2010 pp. 251–268 in Computing Chromosome Conformation, Humana Press.

Hout, M. C., M. H. Papesh, and S. D. Goldinger, 2013 Multidimen- sional scaling. WIREs Cognitive Science 4: 93–103.

Hu, M., K. Deng, Z. Qin, J. Dixon, S. Selvaraj, et al., 2013 Bayesian inference of spatial organizations of chromosomes. PLOS Computational Biology 9: e1002893.

Hunter, J. D., 2007 Matplotlib: A 2D graphics environment. Computing in Science & Engineering 9: 90–95.

Imakaev, M., G. Fudenberg, R. P. McCord, N. Naumova, A. Goloborodko, et al., 2012 Iterative correction of Hi-C data reveals hallmarks of chromosome organization. Nature Methods 9: 999–1003.

Lajoie, B. R., J. Dekker, and N. Kaplan, 2015 The hitchhiker’s guide to Hi-C analysis: Practical guidelines. Methods 72: 65–75.

Lesne, A., J. Riposo, P. Roger, A. Cournac, and J. Mozziconacci, 2014 3D genome reconstruction from chromosomal contacts. Nature Methods 11: 1141–1143.

Li, W., K. Gong, Q. Li, F. Alber, and X. J. Zhou, 2015 Hi-Corrector: a fast, scalable and memory-efficient package for normalizing large-scale Hi-C data. Bioinformatics 31: 960–962.

Lieberman-Aiden, E., N. L. van Berkum, L. Williams, M. Imakaev, T. Ragoczy, et al., 2009 Comprehensive mapping of long range interactions reveals folding principles of the human genome. Science 326: 289–293.

Lyu, H., E. Liu, and Z. Wu, 2019 Comparison of normalization methods for Hi-C data. BioTechniques ahead of print.

MacKay, K. and A. Kusalik, 2019 Computational methods for predicting 3D genomic organization from high-resolution chromosome conformation capture data. Briefings in Functional Genomics Submitted: BFPG–19–0049.

MacKay, K., A. Kusalik, and C. H. Eskivi, 2018 GraphHi-C: graph-based visualization of Hi-C datasets. BMC Research Notes 11: 418.

Mizuguchi, T., J. Barrowman, and S. I. Grewal, 2015 Chromosome domain architecture and dynamic organization of the fission yeast genome. FEBS Letters 589: 2975–2986.

Mizuguchi, T., G. Fudenberg, S. Mehta, J.-M. Belton, N. Taneja, et al., 2014 Cohesin-dependent globules and heterochromatin shape 3D genome architecture in S. pombe. Nature 516: 432–435.

Pedregosa, F., G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, et al., 2011 Scikit-learn: Machine learning in Python. Journal of Machine Learning Research 12: 2825–2830.

Plotly Technologies Inc, 2015 Collaborative data science. Technical report, Plotly Technologies Inc.

Rieber, L. and S. Mahony, 2017 miniMDS: 3D structural inference from high-resolution Hi-C data. Bioinformatics 33: i261–i266.

Rousseau, M., J. Fraser, M. A. Ferraiuolo, J. Dostie, and M. Blanchette, 2011 Three-dimensional modeling of chromatin structure from interaction frequency data using Markov chain Monte Carlo sampling. BMC Bioinformatics 12: 414.

Segal, M. R. and H. L. Bengtsson, 2015 Reconstruction of 3D genome architecture via a two-stage algorithm. BMC Bioinformatics 16: 373.

Sekelja, M., J. Paulsen, and P. Collas, 2016 4d nucleomes in single cells: what can computational modeling reveal about spatial chromatin conformation? Genome Biology 17.

Servant, N., N. Varoquaux, B. R. Lajoie, E. Viara, C.-J. Chen, et al., 2015 HiC-Pro: an optimized and flexible pipeline for Hi-C data processing. Genome Biology 16.

Spill, Y. G., D. Castillo, E. Vidal, and M. A. Marti-Renom, 2019 Binless normalization of Hi-C data provides significant interaction and difference detection independent of resolution. Nature Communications 10: 1938.

Stansfield, J. C., K. G. Cresswell, V. I. Vladimirov, and M. G. Doz- morov, 2018 HiCcompare: an R-package for joint normalization and comparison of Hi-C datasets. BMC Bioinformatics 19.

Tanzawa, H., O. Iwasaki, A. Tanaka, J. R. Capizzi, P. Wickramasing- he, et al., 2010 Mapping of long-range associations throughout the fission yeast genome reveals global genome organization linked to transcriptional regulation. Nucleic Acids Research 38: 8164–8177.

van der Maaten, L., 2009 Learning a parametric embedding by preserving local structure. In Proceedings, Twelfth International Conference on Artificial Intelligence & Statistics (AI-STATS), pp. 384–391, Clearwater, Florida USA, PMLR.

van der Maaten, L., 2014 Accelerating t-SNE using tree-based algorithms. Journal of Machine Learning Research 15: 3231–3245.

van der Maaten, L. and G. Hinton, 2008 Visualizing high dimensional data using t-SNE. Journal of Machine Learning Research 9: 2579–2605.

van der Maaten, L. and G. Hinton, 2012 Visualizing non-metric similarities in multiple maps. Machine Learning 87: 33–55.

Varoquaux, N., F. Aj, W. S. Noble, and J.-P. Vert, 2014 A statistical approach for inferring the 3D structure of the genome. Bioinformatics 30: i226–i33.

Varoquaux, N. and N. Servant, 2019 Iced: fast and memory efficient normalization of contact maps. Journal of Open Source Software 4: 1286.

Wingett, S., P. Ewels, M. Furlan-Magaril, T. Naganjo, S. Schoenfelder, et al., 2015 HiCUP: pipeline for mapping and processing Hi-C data. FL1000Research 4: 1310.

Yang, E.-W. and T. Jiang, 2014 GDNorm: an improved Poisson regression model for reducing biases in Hi-C data. In Algorithms in Bioinformatics, edited by D. Brown and B. Morgenstern, pp. 263–280, Springer Berlin Heidelberg.

Yang, T., J. Liu, L. McMllan, and W. Wang, 2006 A fast approxi-
mation to multidimensional scaling. In *Proceedings of the ECCV Workshop on Computation Intensive Methods for Computer Vision (CIMCV)*, Graz, Austria, IEEE.

Zhang, Z., G. Li, K.-C. Toh, and W.-K. Sung, 2013 3D chromosome modeling with semi-definite programming and Hi-C data. Journal of Computational Biology 20: 831–846.

Zhu, G., W. Deng, H. Hu, R. Ma, S. Zhang, et al., 2018 Reconstructing spatial organizations of chromosomes through manifold learning. Nucleic Acids Research 46: e50.
SUPPLEMENTAL DATA

The following scripts are archived versions of the scripts used to generate the results presented in this manuscript. For the most recent version and/or to report any software problems please see the project homepage at https://github.com/kimmackay/StoHi-C/

Supplementary Script 1: StoHi-C

#!/bin/bash
## Simple shell script that runs both steps of the StoHi-C workflow
## Author: Kimberly MacKay
## Date: December 16, 2019

## the script requires the following command-line inputs: 
## Argument 1: should be the name of the whole-genome contact map
## Argument 2: is the name of the output file for the XYZ coordinates
## Argument 3: is the name of the output file for the distance matrix
## Argument 4: is the filename for the resultant image
## Argument 5: is the filename for the resultant interactive graph (html)

# LICENSE INFORMATION
# This work is licensed under the Creative Commons Attribution-NonCommercial-ShareAlike
# 3.0 Unported License. To view a copy of this license, visit
# http://creativecommons.org/licenses/by-nc-sa/3.0/ or send a letter to Creative Commons,
# PO Box 1866, Mountain View, CA 94042, USA.

echo "running step 1..."
python ./step1/tSNE/run_tSNE.py $1 $2 $3

echo "running step 2..."
python ./step2/plotly_viz.py $2 $4 $5

Supplementary Script 2: StoHi-C Step 1

# StoHi-C Step 1: 3D embedding using tSNE
# This script takes a normalized whole-genome contact map as input and
# embeds the genomic bins in 3D space using TSNE from the sklearn.manifold library
# Argument 1: the file name of the normalized whole-genome contact map
# Argument 2: the output file name for the XYZ coordinates
# Argument 3: the output file name for the distance matrix generated by tSNE

# AUTHOR INFORMATION:
# Kimberly MacKay
# kimberly.mackay@usask.ca
# @mackayka (twitter)

# Authored on April 30, 2019

# LICENSE INFORMATION
# This work is licensed under the Creative Commons Attribution-NonCommercial-ShareAlike
# 3.0 Unported License. To view a copy of this license, visit
# http://creativecommons.org/licenses/by-nc-sa/3.0/ or send a letter to Creative Commons,
# PO Box 1866, Mountain View, CA 94042, USA.

import relevant libraries
import numpy as np
from sklearn.manifold import TSNE
import time
import sys

# define function for reading in data
def populate_matrix(filename, matrix):
    infile = open(filename, "r")
row = 0

for line in infile:
    line = line.rstrip()
    data = line.split("\t")

    col = 0

    # loop through all the elements in the line
    for val in data:
        if val == 'NA':
            dist = 0.0
        elif float(val) == 0.0:
            dist = 0.0
        else:
            dist = 1.0/(float(val)**2)

        matrix[row][col] = dist

        # enforce that matrix[i][j] == matrix[j][i]
        matrix[col][row] = dist

    col = col + 1
    row = row + 1

infile.close()
return matrix

# grab command line arguments
input_file = sys.argv[1]
coord_file = sys.argv[2]
dist_file = sys.argv[3]

# initialize the distance matrix
dist_matrix = np.zeros((1258,1258))
dist_matrix = populate_matrix(input_file, dist_matrix)

# gut check that all the self-self interactions are zero
if sum(dist_matrix.diagonal()) != 0:
    print("WARNING: non-zero elements present in the diagonal")

# run TSNE
# https://scikit-learn.org/stable/modules/generated/sklearn.manifold.TSNE.html
start_time = time.time()

# parameters
# n_components = dimensionality
# perplexity = # of nearest neighbours
# early_exaggeration = determines how "close" nodes will be in the final embedding
# n_iter = maximum number of iterations for the optimization
# method = exact (alternative would be an approximation)
# init = run PCA and use those results as input to tSNE
data_embedded = TSNE(n_components = 3, perplexity=5.0, early_exaggeration=3.0,
                     n_iter=5000, method='exact', init='pca').fit_transform(dist_matrix)

stop_time = time.time()

print("tSNE runtime: \n" + str(stop_time - start_time) + " seconds")

# output embedded data
np.savetxt(coord_file, data_embedded)
Supplementary Script 3: StoHi-C Step 2

## 3D visualization and basic animation of XYZ coordinates from step 1 of StoHi-C

## This script uses plotly to generate a 3D scatter plot
## currently all the parameters are hardcoded for s. pombe data

### Argument 1: the XYZ co-ordinates for each genomic bin generated from step 1
### this file should have the XYZ coords for each bin on a separate line
### each coord should be separated by white space, bins should be in sorted
### numerical order, there shouldn’t be any column or row labels
### Argument 2: the name of the file for the output image
### Argument 2: the name of the file for the output html (interactive graph)

# AUTHOR INFORMATION:
# Kimberly MacKay
# kimberly.mackay@usask.ca
# @mackayk (twitter)

# Authored on Dec. 12, 2019

# LICENSE INFORMATION
# This work is licensed under the Creative Commons Attribution-NonCommercial-ShareAlike 3.0 Unported License. To view a copy of this license, visit http://creativecommons.org/licenses/by-nc-sa/3.0/ or send a letter to Creative Commons, # PO Box 1866, Mountain View, CA 94042, USA.

# import relevant libraries
import sys
import time
import numpy as np
from plotly import graph_objs as go
import chart_studio.plotly as py
import plotly.io as pio

start_time = time.time()

outimagename = sys.argv[2]
outfilename = sys.argv[3]

# read in the data
filename = sys.argv[1]
data_embedded = np.loadtxt(filename)

# generate a figure of the results
fig = go.Figure()

# chr1 – Fission Yeast
fig.add_trace(go.Scatter3d(x=data_embedded[0:558,0],
y=data_embedded[0:558,1],
z=data_embedded[0:558,2],
mode='markers',
opacity=0.5,
name='CHR1'))

# chr2 – Fission Yeast
fig.add_trace(go.Scatter3d(x=data_embedded[558:1012,0],
y=data_embedded[558:1012,1],
z=data_embedded[558:1012,2],
mode='markers',
opacity=0.5,
name='CHR2'))
y=data_embedded[558:1012,1],
z=data_embedded[558:1012,2],
mode='markers',
opacity=0.5,
name="CHR2")

# chr3 – Fission Yeast
fig.add_trace(go.Scatter3d(
x=data_embedded[1012:1258,0],
y=data_embedded[1012:1258,1],
z=data_embedded[1012:1258,2],
mode='markers',
opacity=0.5,
name="CHR3")

# set marker size
fig.update_traces(marker=dict(size=5))

# set axis titles
fig.update_layout(scene = dict(
    xaxis_title='X',
    yaxis_title='Y',
    zaxis_title='Z'))

# set background colours, remove tick labels
fig.update_layout(scene = dict(
    xaxis = dict(
        backgroundcolor="rgb(245,245,245)",
        gridcolor="white",
        showbackground=True,
        zerolinecolor="white",
        showticklabels=False,),
    yaxis = dict(
        backgroundcolor="rgb(230,230,230)",
        gridcolor="white",
        showbackground=True,
        zerolinecolor="white",
        showticklabels=False,),
    zaxis = dict(
        backgroundcolor="rgb(215,215,215)",
        gridcolor="white",
        showbackground=True,
        zerolinecolor="white",
        showticklabels=False,),))

# output results
fig.write_image(outimagename)
plot_url = pio.write_html(fig, file=outfilename, auto_open=False)

stop_time = time.time()
print("Step 2 runtime:
str(stop_time - start_time) + ",seconds")

Supplementary Script 5: MDS

#!/bin/bash
## Simple shell script that runs and visualizes the MDS prediction
## Author: Kimberly MacKay
## Date: December 16, 2019

## the script requires the following command-line inputs:
## Argument 1: should be the name of the whole-genome contact map
## Argument 2: is the name of the output file for the XYZ coordinates
## Argument 3: is the name of the output file for the distance matrix
Supplementary Script 5: MDS Step 1

# MDS Step 1: 3D embedding using MDS
# This script takes a normalized whole-genome contact map as input and
# embeds the genomic bins in 3D space using MDS from the sklearn.manifold library
# Argument 1: the file name of the normalized whole-genome contact map
# Argument 2: the output file name for the XYZ coordinates
# Argument 3: the output file name for the distance matrix generated by tSNE

# AUTHOR INFORMATION:
# Kimberly MacKay
# kimberey.mackay@usask.ca
# @mackayka (twitter)

# Authored on April 30, 2019

# LICENSE INFORMATION
# This work is licensed under the Creative Commons Attribution-NonCommercial-ShareAlike
# 3.0 Unported License. To view a copy of this license, visit
# http://creativecommons.org/licenses/by-nc-sa/3.0/ or send a letter to Creative Commons,
# PO Box 1866, Mountain View, CA 94042, USA.

# import relevant libraries
import numpy as np
from sklearn.manifold import MDS
from sklearn.decomposition import PCA
import time
import sys

# def populate_matrix(filename, matrix):
infile = open(filename, "r")

row = 0

for line in infile:
    line = line.rstrip()
data = line.split("\t")

col = 0

# loop through all the elements in the line
for val in data:
    # need to do a smarter imputation of missing data
    if val == 'NA':
        dist = 0.0
    elif float(val) == 0.0:
        # imput NaNs here
dist = 0.0

else:
    dist = (1.0/(float(val)**2))

matrix[row][col] = dist

# enforce that matrix[i][j] == matrix[j][i]
matrix[col][row] = dist

    col = col + 1
    row = row + 1

infile.close()
return matrix

# grab command line arguments
input_file = sys.argv[1]
coord_file = sys.argv[2]
dist_file = sys.argv[3]

# initialize the distance matrix
dist_matrix = np.zeros((1258,1258))
dist_matrix = populate_matrix(input_file, dist_matrix)

# gut check that all the self-self interactions are zero
if sum(dist_matrix.diagonal()) != 0:
    print("WARNING: non-zero elements present in the diagonal")

# compute the dissimilarity matrix
dissimilarity_matrix = 1 - dist_matrix

# run MDS
#print("running MDS...")
start_time = time.time()

data_embedded = MDS(n_components=3, metric=True, max_iter=5000, dissimilarity='precomputed').fit_transform(dist_matrix)
stop_time = time.time()

print("MDS runtime: " + str(stop_time - start_time) + " seconds")

# output embedded data
np.savetxt(coord_file, data_embedded)

# output distance matrix
np.savetxt(dist_file, dist_matrix)