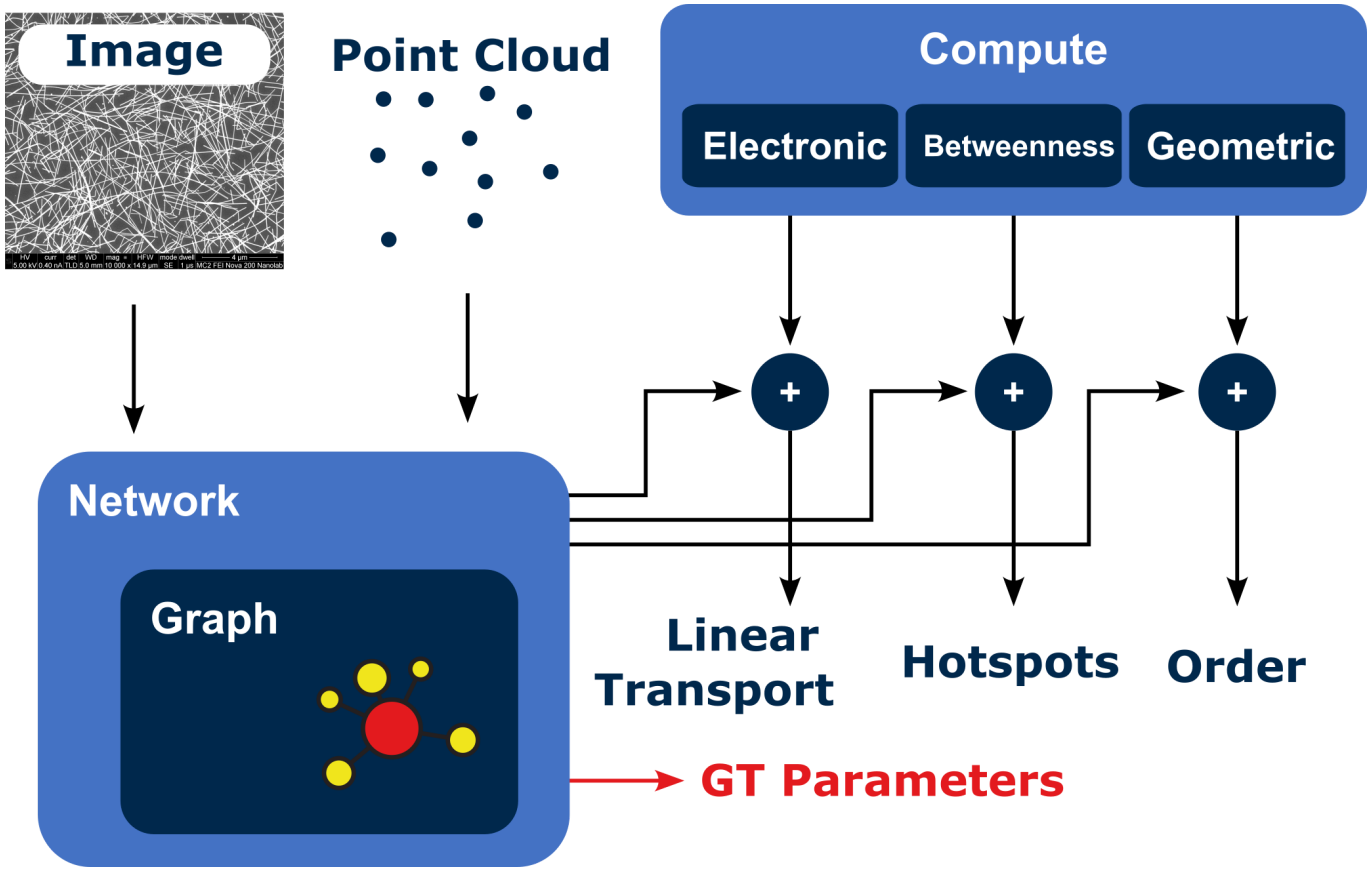

Summary

StructuralGT provides a set of tools for the quantitative analysis of material networks, relying extensively on methods from graph theory (GT). This contribution details the development of an API, flexible filetype, 3D capabilities, and an optimized backend which improves upon the initial release of the StructuralGT GUI application [@Vecchio:2021]. The first purpose of this software is to expose the entire data-to-network workflow via a customizable and extensible API. The second purpose is to offer modular tools for the analysis of the resulting networks. All analysis is can be carried out with the Python API, which makes calls to either fast C libraries or C scripts written specifically for SGT. The advanced user has to option to incorporate their own C scripts using the provided template wrapper. Finally, SGT writes the analyzed graphs to a geometry-preserving filetype allowing for easy visualization of the data with the OVITO desktop application.

Statement of Need

As advanced materials begin to exhibit higher degrees of long-range connectivity patterns, non-random disorder, and hierarchical complexity, a quantitative basis in which to represent their structure is needed. Particularly for nanomaterials, the analasis of network structures has repeatedly been shown to benefit from a GT based representation of their 2D dataREF. As 3D imaging tools become increasingly accessible to experimental communities, tools for their larger and higher-dimensional analyses are needed. Furthermore, the extension of structural representation to property predictions requires tools specific to each system, motivating the development of an extensible user-facing An API also allows for bundling scripts with research publications, which fulfills the growing need for reproducible scientific computing. Collectively,

these requirements necessitate the development of the present contribution whose uses to date include the prediction of charge transport properties of silver nanowire networksREF; the prediction of stress distribution in strut-latticesREF; and the rationalzation of non-monotonic chiroptical properties of complex nanodendrimersREF.

Summary of Features

Graph Extraction and the Network Classes

To most effectively represent the information enshrined in material network data, the `Network` class was developed. Most importantly, the `Network` class can be populated with a `graph` attribute that contains the connectivity information capturing much of the material's structure. Methods of the `Network` class that allow this graph extraction involve image processing, skeletonization, and nearest neighbor finding. Once the appropriate methods have been called, the `graph` attribute of the `Network` class is populated. The graph may be visualized and/or stored in a manner which preserves the undelrying network's geometry.

and the user is free to analyze the graph by making calls to the `igraph`URL??? API. However, because there are geometric features relevant to `In line with Barabasi's terminology, *graph* refers to the mathematical object that can be used to represent real-world networks.REF The `Network` class is used to contain the `graph` subclass, but includes the additional "Data" in this context can refer to ???. The first family of SGT classes are the network classes responsible for converting these data files into a more suitable well-established order parameters.

Network Filetype

While there are many filetypes for storing graphs (e.g. ???), the requirement of preserving geometry in graphs extracted from material networks actually makes filetypes used for molecular simulations a more appropriate choice. For StructuralGT, we have opted for the `.gsd` format, because its compatability with the OVITO toolkit allows us to make rich and dynamic visualizations with both the desktop application and extensive Python API (see Figure for example???). This also makes it compatible with the HOOMD-blue molecular simulation toolkitREF.

Compute Classes

Once the graph attribute of a `Network` object has been populated, it can also be combined with the `Compute` class to carry out analysis of the system. Alternatively, the user may generate a `Network` object with the `graph` attribute already populated by loading from the network filetype mentioned above. In either case, a loaded `Network` can be compbined with compute classes to analyze the network in a number of ways. `Compute` classes allow the user to calculate traditional GT parameters, that are part of standard GT libraries (e.g. with the `Structural` class). But they also contain methods for analyzing geometric features that are typically lost when only looking at a GT based representation of the structure (e.g. orientational and positional order detected with the `Geometric` class). Finally, there are some `Compute` classes that provide estimations of material properties. Examples are the `Electronic` class that provides predictions of uniaxial linear transportREF and `Betweenness` class, which provides predictions of hotspots in physically stressed networks.REF

C++/Python Bindings

Advanced users may wish to write their own C++ scripts and integrate them with StructuralGT. Doing so is made possible using wrappers provided with StructuralGT. These wrappers were used to implement the `Betweenness` and `AverageNodalConnectivity` classes.

Acknowledgements

This research was supported

We would like to acknowledge Drew Vecchio for implementing the initial release of this software and answering any questions we had about it. We would like to acknowledge Joshua Anderson who mentored the implementation of C++/Python bindings, deployment to conda-forge, and software engineering best-practices. We would like to acknowledge the developers of the freud package, whose choice to partition analysis into system and compute classes in their own library inspired the StructuralGT API. Finally we would like to acknowledge the many Glotzer Group and Kotov Lab students who used initial prototypes of StructuralGT and gave feedback on how it could be improved.

Citations

Citations to entries in paper.bib should be in rMarkdown format.

If you want to cite a software repository URL (e.g. something on GitHub without a preferred citation) then you can do it with the example BibTeX entry below for @fidgit.

For a quick reference, the following citation commands can be used:

- `@author:2001` -> "Author et al. (2001)"
- `[@author:2001]` -> "(Author et al., 2001)"
- `[@author1:2001; @author2:2001]` -> "(Author1 et al., 2001; Author2 et al., 2002)"

Figures

Figures can be included like this:  Caption for example figure.\label{fig:example} and referenced from text using \autoref{fig:example}.

Figure sizes can be customized by adding an optional second parameter:  Caption for example figure.{ width=20% }

Acknowledgements

We acknowledge contributions from Brigitta Sipocz, Syrtis Major, and Semyeong Oh, and support from Kathryn Johnston during the genesis of this project.

References