Proceedings of the 2021 Winter Simulation Conference
S. Kim, B. Feng, K. Smith, S. Masoud, Z. Zheng, C. Szabo, and M. Loper, eds.

GraphTrans: A SOFTWARE SYSTEM FOR NETWORK CONVERSIONS FOR SIMULATION, STRUCTURAL ANALYSIS, AND GRAPH OPERATIONS

Henry L. Carscadden
Lucas Machi
Chris J. Kuhlman
Dustin Machi
S. S. Ravi

Biocomplexity Institute and Initiative, University of Virginia
Charlottesville, VA 22904, USA

ABSTRACT

Network representations of socio-physical systems are ubiquitous, examples being social (media) networks and infrastructure networks like power transmission and water systems. The many software tools that analyze and visualize networks, and carry out simulations on them, require different graph formats. Consequently, it is important to develop software for converting graphs that are represented in a given source format into a required representation in a destination format. For network-based computations, graph conversion is a key capability that facilitates interoperability among software tools. This paper describes such a system called *GraphTrans* to convert graphs among different formats. This system is part of a new cyberinfrastructure for network science called *net.science*. We present the *GraphTrans* system design and implementation, results from a performance evaluation, and a case study to demonstrate its utility.

1 INTRODUCTION

1.1 Background and Motivation

There are many uses of socio-technical networks, where nodes or vertices represent entities such as humans, computers, and organizations, and edges represent pairwise interactions between nodes. Social and infrastructure networks are used in a variety of simulations, including the transmission of diseases (e.g., Siettos et al. (2015)), spread of fads and opinions (Weimer et al. 2019), and law enforcement issues like generational influence of incarceration (Lum et al. 2014). Moreover, network analyses are performed on these graphs because explanations for simulation results are often identified through the structure of networks and their properties (e.g., Newman (2003), Rahmandad and Sterman (2008)). Tools to perform simulations, structural analyses, and other operations often require networks in particular formats. However, there are many graph formats, including tailored or custom formats, with no standard representation; for example, the igraph network tool lists 13 formats[1]. Thus, there is a need for tools that convert graphs specified in one format to other formats. (In this paper, we use the terms *graph* and *network* interchangeably; terms *graph conversion* and *graph transformation* are synonymous.)

This motivation can be expanded along several directions. We examine two here. First, different formats for graphs are also helpful in more efficient implementations of graph algorithms. For example, for implementing algorithms for the all pairs shortest path problem, graphs represented as adjacency matrices can exploit parallel matrix multiplication algorithms (Cormen et al. 2009). Thus, for using such algorithms, one may need to convert a graph in a given format into its adjacency matrix representation. In addition,

[1] https://igraph.org/python/doc/tutorial/tutorial.html#igraph-and-the-outside-world

978-1-6654-3311-2/21/$31.00 ©2021 IEEE

as new graph algorithms are developed, it is likely that new graph formats and new graph transformations will be needed for efficiently implementing the algorithms.

Second, software pipelines and workflows are being provided in increasing numbers, e.g., through software gateways and cyberinfrastructures (CIs). See Ahmed et al. (2020) for a summary of these for network science applications. When composing tasks to create workflows that utilize multiple tools, which is a core capability of many CIs, graph transformations are required because these tools, in general, do not use the same graph formats. New CIs and gateways continue to appear, and this trend is likely to only accelerate, as it is consistent with the open-source and open-access software paradigms that have also been in place, and accelerating, over at least the last decade. A key element of the use of these tools, both inside and outside of CIs, is interoperability. For network-based computations, graph conversion is a key element that facilitates interoperability among software tools. We are currently building a CI for general purpose network science, called *net.science* (Ahmed et al. 2020), and graph transformations, which are routinely required in the CI[2], drove the development of *GraphTrans*. For all of the above reasons, it is useful to have tools to automatically convert a graph from one format to another.

1.2 Novelty

We have built a software system to perform graph transformations, from one specified format to another; some of these may require intermediate transformations. The system makes maximum use of transformations provided by existing software libraries. Here, we use SNAP and NetworkX as exemplars. The system also contains a list of formats in different systems that are the same, so that it identifies cases where no transformations are needed. *GraphTrans* is particularly adept at performing conversions across formats from different software systems. These transformations must be reliable and fast because one cannot afford to save every graph in every format, due to storage limitations. This means that in many cases, a graph will be transformed for a particular use, then deleted, and will have to be transformed again when it is needed next. *GraphTrans* can perform these operations with a single invocation, so that it can be used as a component in larger systems. As explained in Section 1.1, the system was built because it is needed for *net.science* CI. However, it can also be used on its own, as a stand-alone system, by individuals and groups.

1.3 Contributions

The source code for *GraphTrans* will be available for download by early 2022. Our other contributions are itemized below.

1. A software system for converting graphs from one format to another. *GraphTrans* is a Python-based software system that converts a graph from its current (source) format F^{src} to a user-specified destination format F^{des}. The system determines a sequence of appropriate transformations that are required to reach the destination format. The sequence, which may include one or more transformations, is determined using a **graph transformation graph** (*GTG*). The *GTG* is a directed graph wherein graph formats are nodes and each directed edge (F^s, F^e) indicates that a graph can be converted *directly* from the (start) format F^s to the (end) format F^e. Thus, given source F^{src} and destination F^{des} formats, a breadth first search (BFS) is performed on the *GTG* to find a shortest path from F^{src} to F^{des}. Note that an edge (F^s, F^e) may represent a graph conversion provided by existing open-source software libraries that are integrated into *GraphTrans*, or graph conversions that we wrote to make transformations across libraries. Currently, the system handles 14 graph formats, so that users can specify 182 ($= 14 \cdot 13$) graph transformations. *GraphTrans*, which can perform transformations on directed and undirected simple graphs and multi-graphs, is described in Section 3.

2. Performance evaluation of the system. We study the performance of the graph transformation code in terms of execution speed. A total of 582 graph transformations are performed on seven networks, which

[2]The initial version of *net.science* was released in June 2021 (URL: https://net.science).

include replicate transformations. These networks range over five (respectively, six) orders of magnitude in number of graph nodes (respectively, graph edges). For the smallest graph tested with 198 nodes and 2,742 edges, graph transformations take on the order of 0.1 seconds. For the largest graph tested with 5.8 million nodes and 129 million edges, graph transformations take on average 5900 seconds. We demonstrate, however, that transformation times can vary significantly (e.g., by four orders of magnitude) for a fixed network, depending on F^{src} and F^{des}. The networks used in the evaluations and the results of the evaluations are presented in Section 4.

3. Case study. We illustrate the practical use of *GraphTrans* by transforming a large social network representation of the 5.8 million person population of Houston, TX, so that it can be used in graph structural analysis computations. The data generated from this case study serve to (i) demonstrate the use and value of *GraphTrans* by presenting properties of networks, and (ii) assess the overheads of transforming graphs by comparing transformation times to the times to produce structural properties. See Section 5.

2 RELATED WORK

Graph analysis libraries and online tools. Software libraries for graph analysis are excellent sources of graph conversion codes. NetworkX (Hagberg et al. 2008), for example, has many graph conversion methods, and we use them. SNAP (Leskovec and Sosič 2016) also has graph transformations that we use. There are other software libraries that we have active plans to integrate, including Gunrock (Wang et al. 2017), a GPU-based software system for graph analysis. Additional libraries include graph-tool NetworKit (Staudt et al. 2014) and igraph (Csardi and Nepusz 2006). There are also online graph conversion tools (e.g., MROCP: Graph Conversion) which allow a user to upload a graph file in some source format and carry out the conversion to a specified destination format; the user may then download the resulting output file. There are several difficulties with this approach, e.g., ability to handle very large graphs, privacy in handling sensitive data, and system extensibility. In *net.science*, we chose to integrate the graph conversion methods in NetworkX and SNAP first because of their very large user bases (at least tens of thousands of downloads each), and because they are complementary: SNAP can handle very large networks and NetworkX has many methods. These large user bases suggest that graph conversions may be useful when using these tools. We demonstrate in Section 3 how our system is extensible and not tied to any library.

Simulators that use networks. *GraphTrans*, our tool for graph conversions, is not focused solely on graph structural analyses. As mentioned in the Introduction, there are many applications for simulations of networked systems. Commensurately, there are many simulation systems that use networks for representing a "population." Among well-known systems are Repast HPC (see Collier and North (2012) and Swarm (Iba 2013)). Surveys and listings of simulators are found in (Railsback et al. 2006). A good overview of agent-based modeling and simulation is provided in (Macal and North 2014); see also (Shen et al. 2019).

Cyberinfrastructures. Section 1.1 motivates the use of our graph conversion tool in CIs. The XSEDE Science Gateway lists more than 50 gateways (https://www.xsede.org/ecosystem/science-gateways); the Science Gateways Community Institute (SGCI) has over 600 gateways (https://catalog.sciencegateways.org/#/home). An admittedly high-level search on the SGCI site using keyword *network science* returns 15 CIs. *GraphTrans* can also be useful in these systems to foster interoperability, and interoperability between gateways.

3 SYSTEM DESCRIPTION

This section discusses several aspects of the system, including the graph formats supported and the algorithm used to obtain the sequence of transformations that will convert a given graph in format G^{src} to the destination format G^{des}. System-level algorithms are also provided.

3.1 Graph Formats

The formats currently supported by the system are: GraphML, GML, MatrixMarket, Edge list (NetworkX, SNAP), adjacency list, GEXF, YAML, Graph6, Multigraph adjacency list, Pajek, GPickle, Sparse6, and SNAP TTable. These represent a large number of popular formats. Section 3.5 addresses how the system handles the addition of a new format.

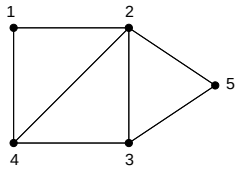


Figure 1: A simple graph *G*.

To illustrate the differences between the descriptions of a graph in two different formats, an example of a small undirected graph *G* is shown in Figure 1. Representations of *G* using SNAP-TTable and GML formats are shown in Figure 2.

```
1    2
1    4
2    4
2    3
2    5
4    3
3    5
```

```
<?xml version="1.0" encoding="UTF-8"?>
<graphml xmlns="http://graphml.graphdrawing.org/xmlns"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://graphml.graphdrawing.org/xmlns
     http://graphml.graphdrawing.org/xmlns/1.0/graphml.xsd">
  <graph id="G" edgedefault="undirected">
    <node id="1"/>
    <node id="2"/>
    <node id="3"/>
    <node id="4"/>
    <node id="5"/>
    <edge source="1" target="4"/>
    <edge source="1" target="2"/>
    <edge source="4" target="2"/>
    <edge source="4" target="3"/>
    <edge source="2" target="3"/>
    <edge source="2" target="5"/>
    <edge source="3" target="5"/>
  </graph>
</graphml>
```

Figure 2: Representations of the graph *G* in Figure 1 using SNAP:TTable format (left) and GML format (right).

3.2 Graph Transformation Graph *GTG*

We use graph libraries, currently SNAP and NetworkX, to perform *direct* graph transformations (F^{src}, F^{des}) where possible. We can extend *GraphTrans* to include other software libraries for other direct transformations. We now address the case where no direct transformation between two formats exists.

Figure 3 shows a portion of the graph transformation graph (*GTG*) used for graph conversions between formats across libraries. This *GTG* is a star graph, with the SNAP TTable format in the center as the hub node. For every graph format F^i and every transformation (F^{src}, F^{des}) that is admissible that does not have a direct graph transformation provided by a network library within *GraphTrans*, the transformation uses TTable as an

intermediate format; the TTable format is designated F^{TT}. A transformation specified by a user, represented as (F^{src}, F^{des}), is broken down into a sequence of two transformations: (F^{src}, F^{TT}) and (F^{TT}, F^{des}). In this way, new formats F^j can be introduced into the system by writing the transformation codes (F^j, F^{TT}) and (F^{TT}, F^j), which we have done in *GraphTrans*. We constructed 24 of the 26 transformations between each of the 13 remaining formats (see Section 3.1) and TTable. Given a specified source format F^{src} and destination format F^{des}, a BFS is performed on the *GTG* to produce a shortest path P_{sd} between the two formats whose transformations are executed (each edge in P_{sd} is a transformation). Note that direct transformations provided by software libraries within *GraphTrans* are also represented as edges in the *GTG*.

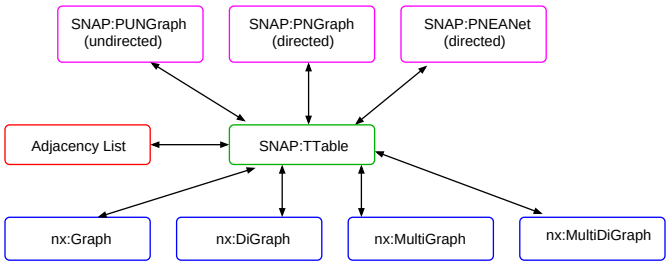


Figure 3: A **graph transformation graph** (*GTG*) for converting graphs between two formats. This representation considers the SNAP and NetworkX graph formats, provided by the respective libraries. Each node $F^i \in V$ in this graph is a graph format, in a rectangle, from the specified graph library. Not all formats are shown. Edges represent transformations. Each edge in the figure between nodes F^i and F^j is displayed as bidirected, signifying two transformations (i.e., two directed edges): from F^i to F^j, $e_{ij} = (F^i, F^j)$, and from F^j to F^i, $e_{ji} = (F^j, F^i)$.

There are several reasons for choosing TTable as the hub format. First, it is the base format in SNAP, and SNAP is known for its ability to handle large networks. Second, and relatedly, the format for SNAP TTable is very similar to that for database tables. For *net.science*, we are considering storing graphs in a database management system (DBMS) to facilitate direct querying of the contents of graph files. TTable provides a consistent format for graph files, for both needs. Third, SNAP provides additional methods for operations on graphs (e.g., forming a union graph from two graphs) and these methods use the TTable format. Based on usage of the system, we are considering storing each graph in TTable format when a transformation through that format in Figure 3 is required. This would reduce the number of transformations required for subsequent graph conversions, and at the same time, provide each graph in the format required for these additional graph operations. The TTable format also reduces hard disk usage because it does not contain syntax per node or per edge that some formats require; see Figure 2 for an example comparison.

3.3 Alternative Solution

The approach described in Section 3.2 is neither the only candidate solution, nor the easiest to implement. Figure 4 provides a more complex *GTG*; the current system uses the simpler *GTG* discussed in Section 3.2. This alternative solution has the advantage that it makes maximum use of graph transformations provided by the SNAP and NetworkX libraries. The downside of this approach is that it is harder to reason about, as the structure is more complicated than that in Figure 3. For example, it is less clear where to insert a new graph format (node) into this graph. Further, as new formats get added over time, the structure of the GTG is likely to become more complex, thus making it more difficult to maintain the software system. Note that both approaches could be used because in each case, a BFS on a GTG, to compute P_{sd}, is possible.

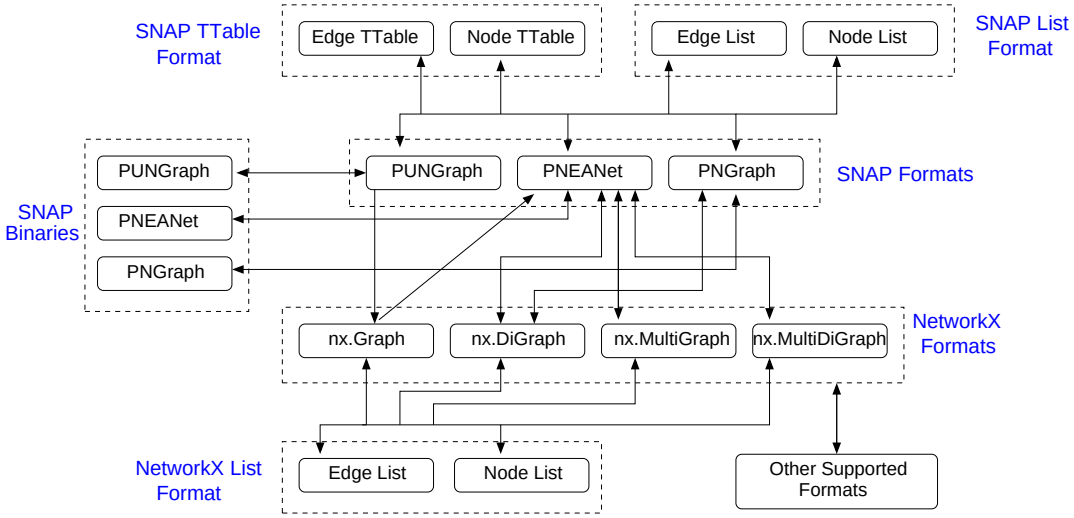


Figure 4: A candidate **graph transformation graph** (*GTG*) for converting graphs between two formats. This representation considers the SNAP and NetworkX graph formats, provided by the respective libraries. Each node $v_i \in V$ (represented by a rectangle) in this graph is a graph format from the specified graph library. Edges represent transformations. For compactness, each edge in the figure between nodes v_i and v_j is displayed as bidirected, but signifies two transformations (i.e., two directed edges): from v_i to v_j, $e_{ij} = (F^i, F^j)$, and from v_j to v_i, $e_{ji} = (F^j, F^i)$.

3.4 Algorithmic Description

Algorithms 1 and 2 provide a formal description of the system. The first algorithm covers the overall steps in the code and the second algorithm focuses on graph transformations that take place across libraries. There are several inputs to each algorithm and the output of *GraphTrans* is the graph in the destination format, denoted G^{des}. In Algorithm 1, after reading the source graph, and the source and destination formats, the code determines whether the two formats are the same (step 3C), and if so, then there is no transformation and the algorithm terminates. Otherwise, a BFS is performed to determine a shortest path from F^{src} to F^{des} in the graph of Figure 3 (step 3D). If the formats are different, and a direct transformation can be effected (step 3E), then this action is taken. The final case is that the specified transformation requires conversions across libraries (step 3F) and this includes a call to Algorithm 2.

3.5 Adding a New Transformation

Consider a new graph format F for which there are no direct graph transformations to and from any other existing format F^i in *GraphTrans*. Then, transformations (F, F^{TT}) and (F^{TT}, F) are effected by subclassing the GraphTransformation class, which has four major methods: read a graph in format F from file, write a graph in format F to file, and effect transformations (F, F^{TT}) and (F^{TT}, F).

4 PERFORMANCE EVALUATION

4.1 Overview

We report results from the performance evaluation of *GraphTrans*. The study included 14 network formats, seven networks, and 582 graph transformation sets. We used the NetworkX and SNAP libraries for selected transformations. Each data point in each plot of this section and the next section is the average of three replicates. Networks used in this study are discussed in the next subsection. These studies as well as those discussed in Section 5 were performed on a high performance computing (HPC) cluster with Dell

Algorithm 1: Graph Transformation

1 **Input:** (1) Name of the file containing the *source* network G^{src}. (2) Whether the graph is directed or undirected. (3) Whether the graph is simple or a multigraph. (4) Whether the graph has attributes (on nodes and/or edges) or not. (5) *source* graph format F^{src} of G^{src}. (6) *destination* graph format F^{des} to which G^{src} is converted. (7) Name of the file to which the *destination* network G^{des} is written. (8) File containing the *GTG* for all the graph transformations provided by the system.

2 **Output:** (1) G^{des} written to file.

3 **Steps:**

A. Read the *GTG* file.

B. Read the *source* and *destination* graph formats F^{src} and F^{des}.

C. **if** $(F^{src} \equiv F^{des})$ **do**:

1. No graph transformation is required because instances of the two formats are the same textually, as follows.
   i. Use the existing *source* format F^{src} for the graph as the *destination* format F^{des}; i.e., set $G^{des} = G^{src}$.
   ii. Return.

D. Perform a breadth first search (BFS) on the *GTG*, from the source graph F^{src} to the destination graph format F^{des}, to determine a directed path graph P_{sd}, from node F^{src} to node F^{des}, i.e., the required sequence of graph transformations.

E. **if** (F^{src} and F^{des} are graph formats from the same library) **do**:

1. Use the particular library's code c_{gt} to perform the graph transformation, as follows.
   i. This means that a software library provides the one-step transformation, represented by the edge (F^{src}, F^{des}).
   ii. Select the provided graph transformation code c_{gt} from the library.
   iii. Generate G^{des} using $G^{des} = c_{gt}(F^{src}, F^{des}, G^{src})$ and write G^{des} to the specified output file.
   iv. Return.

F. **else** // This means that F^{src} and F^{des} are formats from different libraries.

1. Use the path graph P_{sd} obtained from the *GTG*, and Algorithm 2, to produce G^{des} as follows.
   i. Invoke Algorithm 2.
   ii. The return from this call is G^{des}, the graph in the destination format F^{des}.
   iii. Write G^{des} to file.

G. Return.

PowerEdge C6420 2.666 GHz hardware nodes, with 384 GB RAM and 40 cores per node. Each core in a node is an Intel Xeon Gold 6148, 2.40 GHz processor with 1280 KiB L1 cache, 20 MiB L2 cache, and 27 MiB L3 cache. (*GraphTrans* is a serial code and uses only one compute node per execution.)

4.2 Networks

Social networks used in our performance and case studies are shown in Table 1. These networks were selected to cover five orders of magnitude in numbers n of nodes, and six orders of magnitude in numbers m of edges. Selected structural properties of these networks are also provided; these values were generated using *GraphTrans* and the codes in the *net.science* CI (Ahmed et al. 2020) for network science, which contains the structural analysis methods in SNAP (Leskovec and Sosič 2016) and NetworkX (Hagberg et al. 2008). The smallest networks come predominantly from (Jure Leskovec and Andrej Krevl 2014). The larger three social contact networks were generated according to the procedures in Barrett et al. (2009).

Algorithm 2: Multi-Format Graph Conversions, i.e., Conversions Across Libraries

1 **Input:** (1) Name of the file containing the *source* network G^{src}. (2) Whether the graph is directed or undirected. (3) Whether the graph is simple or a multigraph. (4) Whether the graph has attributes (on nodes and/or edges) or not. (5) *source* graph format F^{src} of G^{src}. (6) *destination* graph format F^{des} to which G^{src} is converted. (7) the path graph P_{sd}.

2 **Output:** The graph G^{des} in the destination format F^{des}.

3 **Steps:**

A. Read G^{src} into memory.

B. Assign the in-memory representation of G^{src} to G.

C. Traverse the path graph P_{sd}, from node F^{src} to node F^{des}, in transforming the graph G, as follows.

1. Set $F^s = F^{src}$.
2. **for each** (edge in P_{sd}) **do**:
   i. The edge is (F^s, F^e). G is in format F^s by construction.
   ii. Transform the graph from format F^s to F^e. So now the graph G is in format F^e.
   iii. Set $F^s = F^e$.
3. By construction, the graph G is now in format F^{des}, i.e., we have G^{des}.

D. Return the graph G^{des}, which is in format F^{des}.

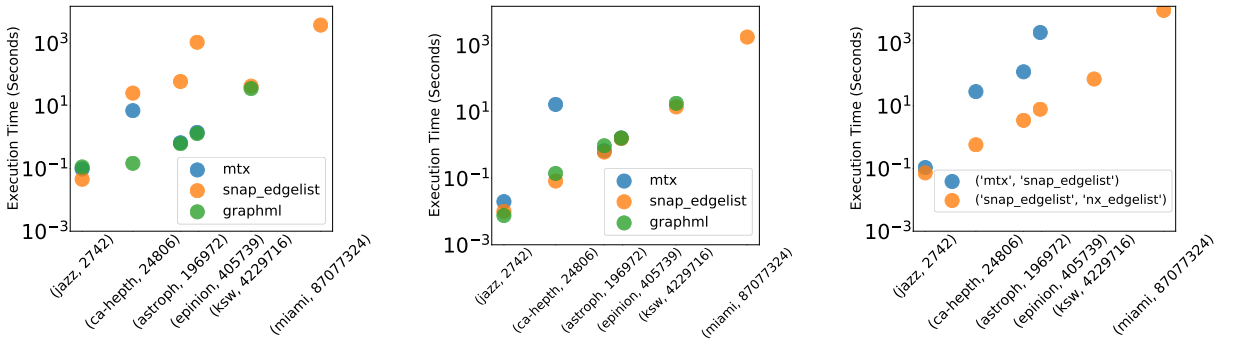
Table 1: Social networks used in performance evaluation and case study. If there are multiple connected components in a graph, we use only the giant component. Here, n and m are numbers of vertices and edges, respectively, in the giant component; d_{ave} and d_{max} are average and maximum degrees; k_{max} is maximum kcore; c_{ave} is average clustering coefficient; and Δ is graph diameter. Jazz is a network of musicians, Ca-Hepth and Ca-Astroph are collaboration networks, Epinions represents trust on Epinions, and VA-Beach, Miami, and Houston are synthetic population networks. The networks for Miami and Houston are the social contact networks among individuals in these cities for a normative Wednesday, among the most active week days.

Network	**Edge Multiplicity**	**Edge Direction**	n	m	d_{ave}	d_{max}	k_{max}	c_{ave}	Δ
Jazz	simple	undirected	198	2,742	27.7	100	29	0.617	6
Ca-Hepth	simple	undirected	8,638	24,806	5.74	65	31	0.482	18
Ca-Astroph	simple	undirected	17,903	196,972	22.0	504	56	0.633	14
Epinions	simple	undirected	75,877	405,739	10.7	3044	67	0.138	14
VA-Beach	simple	directed	167,722	4,229,716	50.4	779	760	0.186	10
Miami, FL	simple	undirected	4,894,150	87,077,324	35.6	656	47	0.092	11
Houston, TX	simple	undirected	5,841,222	129,310,137	44.3	706	51	0.073	12

4.3 Execution Speed

Figure 5 shows three plots of time used by transformations. Since TTable is the hub format in Figure 3, the plots show data for the cases: (i) TTable is the destination format (source formats in legend), (ii) TTable is the source format (destination formats in legend), and (iii) transformations are *through* TTable in going from F^{src} to F^{des} per legend. Thus, the transformations in the first two plots are single-hop, and those in the last plot use two hops. In Figures 5a and 5b, the blue points for MatrixMarket (label "mtx") are underneath the orange points for the SNAP edgelist format for the Ca-Astroph and larger networks. We are currently investigating why execution times for MatrixMarket and SNAP edgelist execution times decrease, as the graph size increases (from Epinions to VA-Beach) in Figure 5a; we have run replicates and the behavior

is repeatable. The plots demonstrate that execution times for the same transformation can vary by four to five orders of magnitude across networks.



(a) transformations to F^{des} =TTable (b) transformations from F^{src} =TTable (c) transformations *through* format TTable

Figure 5: Performance data in terms of execution times for graph transformations for each graph (on the x-axis). The three plots have different classes of transformations, as follows. (a) Transformations times from the identified graph format F^{src} in the legend to F^{des} = TTable. (b) Transformation times from F^{src} = TTable to the formats F^{des} identified in legend. (c) Transformation times from the specified source F^{src} to destination F^{des} formats, where each transformation goes through TTable; thus, these transformations are 2-hop.

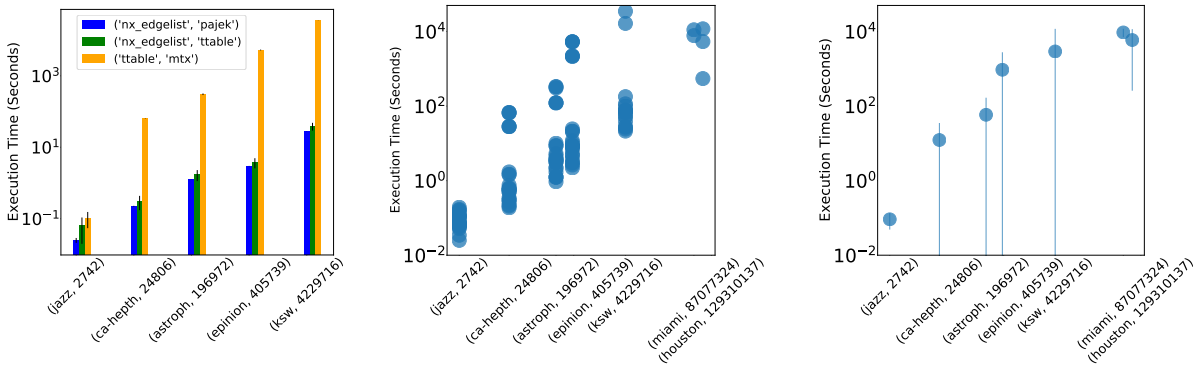
Figure 6a shows three transformations for each graph along the x-axis. The three transformations are chosen from the data to represent lesser, intermediate, and greater execution times. The large execution times involve the MatrixMarket format (labeled "mtx," orange bars).

Figure 6b shows data for a total of 119 transformations over the seven networks. (In this plot, unlike the other plots, data points are individual measured times and do not represent the average of multiple replicates.) The purpose is to show the range of transformation times possible for each network, e.g., Epinions has times over almost five orders of magnitude. For each of the networks Ca-Hepth, Ca-Astroph, Epinions, and VA-Beach, the two data points with the greatest execution times are transformations from edge lists to the MatrixMarket format.

Table 2 provides a representative breakdown of times for major steps of the transformation process. The BFS traversal of the *GTG* is fast, as expected, but reading and writing a graph can consume significant time. Moreover, differences in reading and writing times can be significantly different even though it is the same graph (compare the contents of graph files in Figure 2).

Table 2: Representative measured times for selected tasks and the total execution time for graph transformation for the Houston network from format NetworkX Edgelist to format SNAP Edgelist.

Task	**Task Execution Time(s)**
Perform BFS on *GTG* to obtain P_{sd}.	0.000017
Read graph in format F^{src} from file.	2721.9
Write graph in format F^{des} to file.	411.8
Perform in-memory transformation from F^{src} to F^{des}.	4833.0
Total code execution time.	7966.7



(a) All transformation times, error bars (b) All transformation times, scatter (c) All transformation times, error bars

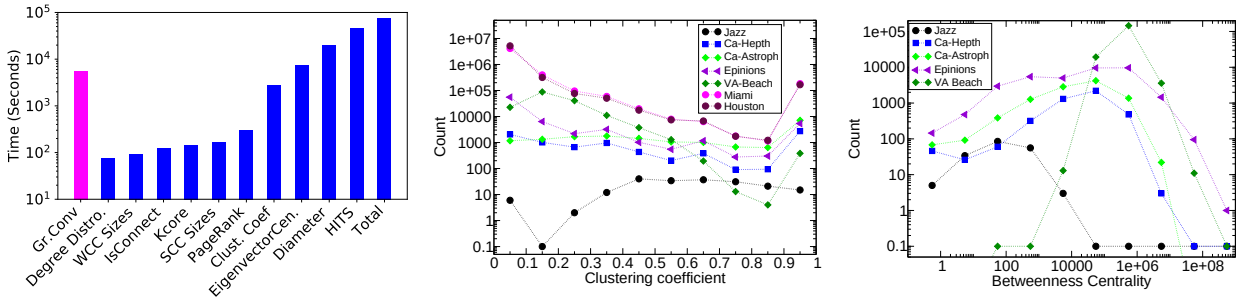
Figure 6: (a) Performance data in terms of execution time across multiple graph transformations for five graphs (on the x-axis). The three transformation types are chosen based on the following: the first transformation is among the fastest across all networks; the second transformation is a representative transformation that has intermediate execution times; and the third transformation is among the most time consuming across all networks. Bars have black lines indicated $\pm$ one standard deviation. These are all "one-hop" transformations. The directed edge (F^{src}, F^{des}) for each transformation per network is in the legend. (b) Performance data in terms of execution times for all networks. Each x-axis value is one network, with number of edges also listed. For each network, several data points are given that represent times for various transformations; each data point corresponds to one of 119 transformations. The purpose is to identify the ranges in times that transformations can take, depending on format. (c) The data point for each network represents the mean of all transformations in plot (b), and error bars represent $\pm$ one standard deviation on the log scale.

5 CASE STUDY

The primary purpose of our case study is to illustrate the practical use of *GraphTrans*. A network representation of the pairwise interactions among the 5.8 million people in Houston, TX is specified in a NetworkX format. The goal is to generate structural analysis parameters of the network, so that these can be used to reason about simulation results; see for example (Kuhlman et al. 2015). The structural analysis methods in SNAP and NetworkX, and "home-grown" codes, are used. For SNAP, the graph needs to be transformed to the PUNGraph format. Structural properties are generated and code execution times for these methods are recorded.

Figure 7a provides the timing data, where the first bar (magenta) is the graph transformation time ("Gr. Conv"). This transformation time is greater than the time to compute several structural properties (the total time for all structural analyses is the last bar). These results can be understood as follows. First, the graph transformation requires two hops, through TTable in Figure 3, and second, the reading and writing of a network consumes considerable time (see Table 2).

Two plots showing the computed structural properties are shown: Figure 7b contains clustering coefficient distributions for all networks and Figure 7c depicts betweenness centrality distributions for five networks. Plots illustrate characteristic curves for each property, e.g., clustering coefficient distributions often have a concave-upward shape, with larger numbers of nodes with near-zero and near-one clustering coefficient values. These data serve two purposes: (i) they demonstrate the use and value of *GraphTrans*, and (ii) the timing comparisons give a sense of the "overhead" associated with transforming graphs.



(a) Timing data, Houston network

(b) Clustering coeff. distributions

(c) Betweenness centr. distribution

Figure 7: Various results enabled by graph transformations. (a) Timing data for the Houston network. The magenta bar is the time to convert it from NetworkX graph format to SNAP PUNGraph edge list. The remaining bars are the times to compute various structural properties of the Houston network. (b) Clustering coefficient distributions for all networks of this study (the time to generate these data for the Houston network is given in the "Clust.Coef." blue bar in (a)). (c) Betweenness centrality distributions for five of the seven networks. (This computation is a very expensive in terms of time; results were not available for the largest two networks even after 296 hours (2 weeks) of computation on an HPC cluster).

6 SUMMARY

The *GraphTrans* system for large graph conversions has been presented, with contributions of our work in Section 1.3. The system is described, evaluated, and used in a case study in Sections 3, 4, and 5, respectively. Future enhancements to the system will include adding conversion codes for other graph representation formats and identifying other base formats (similar to SNAP TTable used in Figure 3) that can help in making certain common transformations run faster.

ACKNOWLEDGMENTS

We thank the anonymous reviewers for their helpful comments. We thank our colleagues at NSSAC and Research Computing at The University of Virginia for providing computational resources and technical support. This work has been partially supported by University of Virginia Strategic Investment Fund award number SIF160, NSF Grant OAC-1916805 (CINES), and NSF Grant CMMI-1916670 (CRISP 2.0).

REFERENCES

Ahmed, N. K., R. A. Alo, C. T. Amelink et al. 2020. "net.science: A Cyberinfrastructure for Sustained Innovation in Network Science and Engineering". In *Gateway Conference*. Oct. 19-23, Virtual.

Barrett, C. L., R. J. Beckman et al. 2009. "Generation and Analysis of Large Synthetic Social Contact Networks". In *Winter Simulation Conference (WSC)*, edited by M. D. Rossetti, R. R. Hill, B. Johansson, A. Dunkin, and R. G. Ingalls, 1003–1014. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc. Dec. 13-16, Austin, TX, USA.

Collier, N., and M. North. 2012. "Parallel Agent-Based Simulation with Repast for High Performance Computing". *Simulation* 89(10):1215–1235.

Cormen, T. H., C. E. Leiserson, R. L. Rivest, and C. Stein. 2009. *Introduction to Algorithms*. Cambridge, MA: MIT Press and McGraw-Hill.

Csardi, G., and T. Nepusz. 2006. "The igraph Software Package for Complex Network Research". *InterJournal* Complex Systems.

Hagberg, A. A., D. A. Schult, and P. J. Swart. 2008. "Exploring Network Structure, Dynamics, and Function Using NetworkX". In *7th Python in Science Conference (SciPy2008)*, 11–15.

Iba, H. 2013. *Agent-Based Modeling and Simulation with Swarm*. Chapman & Hall/CRC. New York, NY.

Kuhlman, C. J., V. S. A. Kumar et al. 2015. "Inhibiting Diffusion of Complex Contagions in Social Networks: Theoretical and Experimental Results". *DMKD* 29:423–465.

Jure Leskovec and Andrej Krevl 2014, Jun. "SNAP Datasets: Stanford Large Network Dataset Collection". http://snap.stanford.edu/data. Accessed Oct. 14, 2020.

Leskovec, J., and R. Sosič. 2016. "SNAP: A General-Purpose Network Analysis and Graph-Mining Library". *ACM Transactions on Intelligent Systems and Technology (TIST)* 8(1):1.

Lum, K. et al. 2014. "The Contagious Nature of Imprisonment: An Agent-Based Model to Explain Racial Disparities in Incarceration Rates". *Journal of The Royal Society Interface* 11(98).

Macal, C., and M. North. 2014. "Introductory Tutorial: Agent-Based Modeling and Simulation". In *Winter Simulation Conference (WSC)*, edited by A. Tolk, S. Y. Diallo, I. O. Ryzhov, L. Yilmaz, S. Buckley, and J. A. Miller, 6–20. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc. Dec. 7-10, Savannah, GA, USA.

Newman, M. E. J. 2003. "The Structure and Function of Complex Networks". *SIAM Review* 45:167–256.

Rahmandad, H., and J. Sterman. 2008. "Heterogeneity and Network Structure in the Dynamics of Diffusion: Comparing Agent-Based and Differential Equation Models". *Management Science* 54:998–1014.

Railsback, S. F., S. L. Lytinen, and S. K. Jackson. 2006. "Agent-Based Simulation Platforms: Review and Development Recommendations". *SIMULATION* 82(9):609–623.

Shen, W., R. Achar, and C. V. Lopes. 2019. "A Simulation Analysis of Large Contests with Thresholding Agents". In *Winter Simulation Conference (WSC)*, edited by N. Mustafee, K.-H. Bae, S. Lazarova-Molnar, M. Rabe, P. H. C. Szabo, and Y.-J. Son, 181–192. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc. Dec. 8-11, National Harbor, MD, USA.

Siettos, C. et al. 2015. "Modeling the 2014 Ebola Virus Epidemic–Agent-Based Simulations, Temporal Analysis and Future Predictions for Liberia and Sierra Leone". *PLOS Currents Outbreaks*:22 pages.

Staudt, C., A. Sazonovs, and H. Meyerhenke. 2014. "NetworKit: A Tool Suite for Large-Scale Complex Network Analysis". *Network Science* 4(4):508–530.

Wang, Y., Y. Pan et al. 2017. "Gunrock: GPU Graph Analytics". *ACM Trans. on Parallel Computing* 4(1).

Weimer, C. W., J. O. Miller et al. 2019. "Agent Scheduling in Opinion Dynamics: A Taxonomy and Comparison Using Generalized Models". *Journal of Artificial Societies and Social Simulation (JASSS)*:19 pages.

AUTHOR BIOGRAPHIES

HENRY CARSCADDEN is an undergraduate student in the Computer Science Department at the University of Virginia (UVA). His email address is hlc5v@virginia.edu.

LUCAS MACHI is an undergraduate student at New River Community College and is a member of the Biocomplexity Institute and Initiative (BII) of the UVA. His email address is lhm4v@virginia.edu.

CHRIS J. KUHLMAN, DUSTIN MACHI, S. S. RAVI are faculty and staff at the BII of the UVA. Their email addresses are [cjk8gx], [dm8qs], [ssr6nh]@virginia.edu.