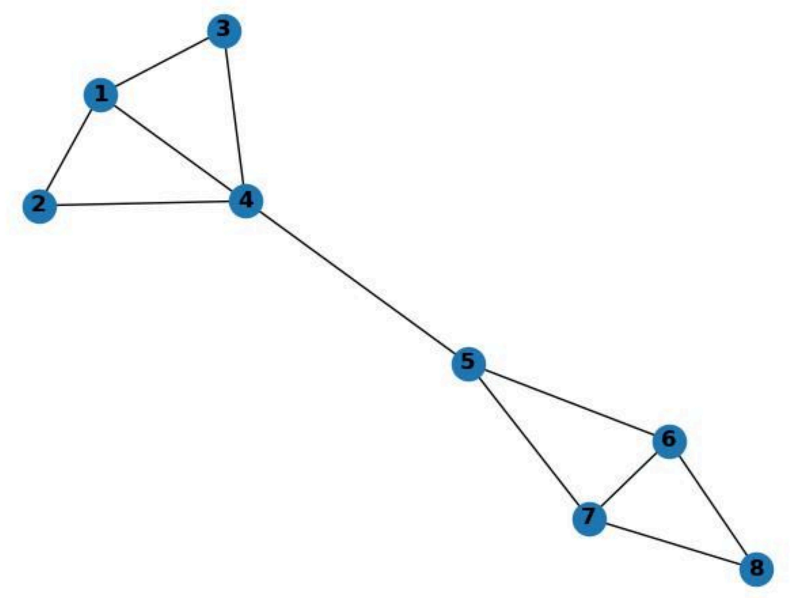
Improving Entity Resolution with Graph Data Science

Graph Data Science is a cutting-edge field which applies Graph Theory to solve many data-related problems in the business world. Traditionally, graph theory is used heavily in computer science and a subfield of math called Probability, for modeling series of events. In Data Surge's case, we employ graph analytics to improve *entity resolution* for clients: matching records from multiple data sources to ensure joins are perfectly clean and each ID is distinguished without error. Error-free databases ensure businesspeople and customers use products reliably, quickly and without confusion. Entity Resolution keeps software and analytics running smooth!



A custom graph created with the Python package NetworkX

In the Entity Resolution world, the work of graph algorithms is generally drawn into two tasks: linking and grouping. *Linking* determines whether two individual nodes are connected, whereas *grouping* determines if a set of linked nodes forms a *community*. The graph algorithms typically used for entity resolution are called "community detection" algorithms. Ergo, in graph terms, Entity Resolution is about creating error-free communities!

At its core, Graph Data Science (GDS) is about finding connections between points (or *nodes*) in a network, which makes it ideal for Entity Resolution; the goal of entity resolution is to evaluate how strongly connected (or disconnected) two records are to each other and decide if they're part of the same entity.

Identifying a Business Problem for GDS

Data Surge recently had a client with billions of records pertaining to people, and needed a system which reliably and cleanly matched a group of records to a single person. The client already had a model employing arbitrary, rule-based matching and machine learning, but still combined some records erroneously; multiple people's records were matched to the same ID. By adding Graph Data Science methods to their technology stack, we were able to eliminate the overwhelming majority of erroneous matches in our test data. It is often best to employ GDS methods **in combination with** rules based on your domain expertise and/or feed the mostly-clean data through a traditional machine learning model. You may then deploy a graph algorithm and feed it multiple features.

Deploying Graph Tools

One of the most popular (and simplest) algorithms for GDS is Label Propagation. Its simplicity makes it widely available in popular Python packages including Neo4J, Scikit-Learn and in GraphFrames for PySpark. With Label Propagation, a few points in a network are randomly assigned labels, then, as the algorithm iterates, it assigns each node the most common label of its immediate neighbors. But many graph algorithms are easily understood and widely available among Graph Analytics tools!

When preprocessing your data for a graph project, your labels will be the node identifiers and graph algorithms will use one or more features to determine connections among the nodes. GDS packages will generally offer the ability to search your data for communities based on a feature or common *motif* (i.e. nodes with the first name "John" connected to nodes with the last name "Doe"). You may also query nodes according to their "degree" (the number of other nodes they are connected to) in order to find especially isolated or overloaded nodes, which may identify weaknesses in your GDS deployment. There's a lot of useful tools for a data professional here!

An example of GDS for Entity Resolution: Transitive Matching

Remember the graph we saw earlier? It came from a simplified dataset of people's records that I created. In our data, we have two individuals (Stephen Graphman and Stephanie Garrison) erroneously assumed to be one person by an inferior model which doesn't employ graph methods.

	First Name	Last Name	Home State	Phone #	AssignedSystemID
1	Stephen	Graphman	Maryland	(555) 555-5555	12345
2	Stephen	Graphman	Maryland	555-5555	12345
3	Steve	Graphman	MD	(555) 555-5555	12345
4	Stephen	G.	MD	(blank)	12345
5	Steph	G.	MN	(blank)	12345
6	Steph	Garrison	MN	(blank)	12345
7	Stephanie	Garrison	MN	(444) 444-4444	12345
8	Stephanie	Garrison	Minnesota	(444) 444-4444	12345

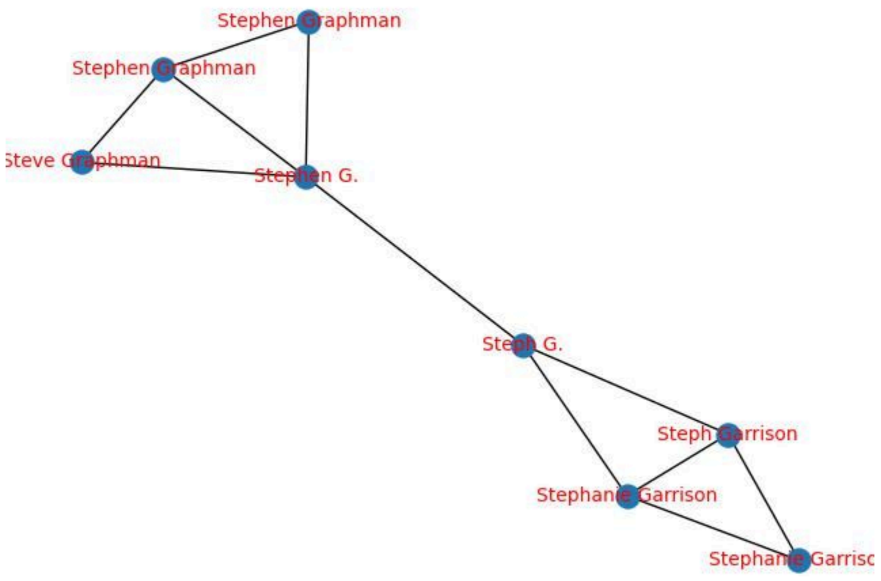
Assume our system's current entity resolution system saw these records and gave them all the same **AssignedSystemID** of 12345. At first, it correctly assigned records 1-4 to one community (the Stephen Graphman community"), then correctly created a community from records 5-8 (the "Stephanie Garrison community"). But when it encountered records 4 and 5, with so many similar fields, it erroneously assumed the groups to be the same community.

Traditional ML's Flaw and Where GDS Offers Improvement

This error is unfortunately common among ML models attempting entity resolution: it measured the string similarity between records 4 and 5, got a high score, and therefore assumed 4 and 5 referred to the same person. This mistake is nicknamed the "transitive matching error" because it resembles the transitive property in mathematics: `if a = b and b = c, then a = c`. The model assumed that, because one Stephen Graphman record resembled one Stephanie Garrison record, all records have the same identity (think of records 1-4 as "a", 4-5 as "b", and 5-8 as "c").

Many machine learning models would create a "transitive" link for two records like this because both have the same last name and phone number, similar state abbreviations, and similar first names. Even the similar phone numbers can influence a lot of machine learning pipelines which use word distance to couple these people together! With graph analytics, though, we can examine each record's relationship to the nodes *around* it to properly split the data into two communities.

Let's see our graph again. In NetworkX, we can change the label of each node to any feature we'd like, so I'm picking the name here.

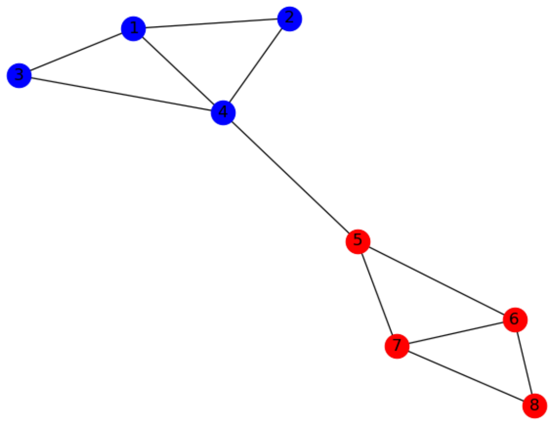


Communities victim to transitive matching will often resemble a “barbell” with one dense cluster on either side joined by a lone connector in the middle.

Graph tools can be fed the features of the model to determine links between records (or nodes) just like any machine learning tool out there. But what differentiates a graph tool from a traditional machine learning model is how it examines the relationships of the node to its surrounding nodes. Some aspects of the nodes it would examine include:

- the number of connections from one node to other nodes
- *which* nodes are pointing to a particular node (do a lot of nodes point to node A, which then points to node B? In that case, node A may also be influential or relevant to this community)
- the number of nodes one must traverse to reach another node
- feature similarities shared by all nodes in a community
- how a community changes, on average, when one node is removed or added to a community
- empirically, how often does a random walk on this graph from point A wind up at point B (or its neighbors)?

These different methodologies result in clearer determination of communities. So let’s run LPA on our sample data and see what it gives us:



Blue is the Stephen Graphman community, red is Stephanie Garrison. Results from NetworkX's label_propagation_communities function.

Because LPA was able to look at the number of nodes *pointing to* each node, it was able to correctly identify two separate communities here! DataSurge leveraged these same techniques to deliver significant accuracy gains for our client.

Potential Pitfalls of Graph Algorithms

No cutting-edge algorithm is going to be a panacea for your data problems, and GDS is not without its weaknesses. LPA may create more communities than necessary, leading to "false negatives" in the results (erroneously splitting records into multiple identities). But, in Data Surge's case, the client was fine with an occasional false negative because there are multiple fields to search for a record; it is more convenient for them to split one person into multiple identities because they can always check multiple fields to make sure they have all of one person's records. In other words, it's not okay for multiple people to be attached to the same ID, but it's fine for one person to be attached to multiple IDs (on rare occasion). Needs vary from project to project; to adjust the rate of false negatives, users can lower or raise the requirements for joining nodes into a community, or determining if two nodes are connected.

Individually, most graph algorithms are computationally efficient. But with highly complex networks looking at multiple features, or if you change the parameters of the algorithm to run for hundreds of iterations, they can be costly.

The World of Graph Analytics

Graph Data Science has solved problems in several topics related to Entity Resolution as well, including:

- Creating customer segments to design marketing campaigns and personalized product experiences
- Analyzing relationships between social media accounts to detect spambots spreading misinformation
- Finding similar characteristics among bank accounts worldwide to identify fraudsters routing laundered money back to themselves or associates
- Identifying highly contagious patients so they may be isolated and limit the spread of illness

Ultimately, the effectiveness of Graph Data Science is limited only by your ability to frame a business problem in Graph Theory terms, and format your data to work with GDS tools. Data Surge has developed a solutions accelerator for solving entity resolution issues using graph analytics, wherein we apply our GDS expertise to quickly and effectively deliver cleaner databases for clients.