S3G2: a Scalable Structure-correlated Social Graph Generator

Minh-Duc Pham[1] Peter Boncz[1] Orri Erling[2]

[1] CWI, Amsterdam, Netherlands,
duc@cwi.nl, boncz@cwi.nl,
[2] OpenLink Software, Burlington, U.S.A
oerling@openlinksw.com

Abstract. Benchmarking graph-oriented database workloads and graph-oriented database systems are increasingly becoming relevant in analytical Big Data tasks, such as social network analysis. In graph data, structure is not mainly found inside the nodes, but especially in the way nodes happen to be connected, i.e. structural correlations. Because such structural correlations determine join fan-outs experienced by graph analysis algorithms and graph query executors, they are an essential, yet typically neglected, ingredient of synthetic graph generators. To address this, we present S3G2: a Scalable Structure-correlated Social Graph Generator. This graph generator creates a synthetic social graph, containing non-uniform value distributions and structural correlations, and is intended as a testbed for scalable graph analysis algorithms and graph database systems. We generalize the problem to decompose correlated graph generation in multiple passes that each focus on one so-called *correlation dimension*; each of which can be mapped to a MapReduce task. We show that using S3G2 can generate social graphs that (i) share well-known graph connectivity characteristics typically found in real social graphs (ii) contain certain plausible structural correlations that influence the performance of graph analysis algorithms and queries, and (iii) can be quickly generated at huge sizes on common cluster hardware.

1 Introduction

Data in real life is correlated; e.g. people living in Germany have a different distribution in names than people in Italy (location), and people who went to the same school in the same period have a much higher probability to be friends in a social network. Such correlations can strongly influence the intermediate result sizes of query plans, the effectiveness of indexing strategies, and cause absence or presence of locality in data access patterns. Regarding intermediate result sizes of selections, consider:

```
SELECT personID FROM person
WHERE firstName = 'Joachim' AND addressCountry = 'Germany'
```

Query optimizers commonly use the *independence assumption* for estimating the result size of conjunctive predicates, by multiplying the estimates for the individual predicates. This would underestimate this result size, since Joachim is more common in Germany than in most other countries; similar would happen e.g. when querying for firstName 'Cesare' from 'Italy'. Overestimation can also easily happen, if we would query for 'Cesare' from 'Germany' or 'Joachim' from 'Italy' (i.e. *anti-correlation*).

This correlation problem has been recognized in relational database systems as relevant, and some work exist to detect correlated properties inside the same table [12]. Still, employing techniques for the detection of correlation is hardly mainstream in relational database management, and this is even more so when we start considering correlations between predicates that are separated by joins. Consider for instance the DBLP example of co-authorship of papers that counts the amount of authors that have published both in TODS and in the VLDB Journal:

```
SELECT COUNT(*)
FROM paper pa1 JOIN journal jn1 ON pa1.journal = jn1.ID
     paper pa2 JOIN journal jn2 ON pa2.journal = jn2.ID
WHERE pa1.author = pa2.author AND
      jn1.name = 'TODS' AND jn2.name = 'VLDB Journal'
```

The above query is likely to have a larger result size than a query that substitutes 'TODS' for 'Bioinformatics', even though Bioinformatics is a much larger publication than TODS. The underlying observation is that database researchers are likely to co-publish in TODS and The VLDB Journal, but are much less likely to do cross-disciplinary work. For database technology, this example poses (i) a challenge to the optimizer to adjust the estimated join hit ratio of `pa1.author = pa2.author` downwards or upwards depending on other (selection or join) predicates in the query (ii) provide indexing support that can accelerate this query: the anti-correlated query (Bioinformatics and The VLDB Journal) has a very small result size and thus could theoretically be answered very quickly. However, just employing standard join indices will generate a large intermediate result for the Bioinformatics sub-plan containing all Bioinformatics authors, of which only a minute fraction is actually useful for the final answer.

Summarizing, correlated predicates are still a little explored research frontier in database research, and such queries are generally not well-supported yet in mature relational systems. This holds still more strongly in the emerging class of graph database systems, where we argue the need for correlation-awareness in query processing is even higher.

In the particular case of RDF, its grapg data model is expressly chosen to work without need for an explicit schema, such that graph datasets get stored as one big pile of edges (in particular, subject-property-object "triples"). Here we see a dualism between structure and correlation: in the relational model, certain structure is explicit in the schema, whereas in RDF such structure only re-surfaces as structural *correlation*. That is, it will turn out a journal paper (subject) always happens to have one title property, one issue property, one journal name, etc; and that these properties exclusively occur in connection to journal issues.

The extreme flexibility of RDF systems in the data they can store, thus poses significant challenge to SPARQL query optimizers, as they need to understand such correlations to get the planning of even basic queries right. Other graph database systems which use a richer data model, where nodes have a declared structure, suffer less from this problem. Still, when considering that graph analysis queries often involve a combination of (property) value constraints and structural constraints (pattern matching), it is clear that correlations between the structure of the graph and the values in them will strongly affect the performance systems and algorithms. As such, we argue that for *benchmarking* graph data analysis systems and algorithms, it would be very worthwhile if a data generator could generate *synthetic graphs* which such *correlated structure*.

To our knowledge, there exists no solution for generating a scalable random graph with value and structure correlations. Existing literature on random graph generation [3, 9, 5, 7] either does not consider node properties at all or ignores correlations between them.

In this paper, we describe the Scalable Structure-correlated Social Graph Generator (S3G2), and its underlying generic conceptual correlated graph generation framework. This framework organizes data generation in multiple phases that each center around a *correlation dimension*. In

the case of our social graph use case, these dimensions are (i) location and (ii) personal interests. The data generation process is further instrumented by *correlation rules* that tell how already generated data influences the generation of additional data. A graph generator generates new nodes (with property values), and edges between these nodes and existing nodes. The probability to choose a certain value from a dictionary, or the probability to connect two nodes with an edge are thus influenced according to these correlation rules, by existing data values. For instance, the birth location of a person influences probability distribution of the firstName and school dictionaries. As another example, the probabilioty to create a friendship edge is influenced by agreement on birth-year and school properties of two person nodes.

A practical challenge in S3G2 is that a naive approach to correlated graph generation would continuously access possibly any nodes and any edge in order to make decisions regarding the generation of a next node or edge. For generating graphs of a size that exceeds RAM, such a naive algorithm would grind down due to expensive random I/O. To address this challenge, we designed a S3G2 graph generation algorithm following the MapReduce paradigm. Each pass along one correlation dimension is a Map phase in which data is generated, followed by a Reduce phase that sorts the data along correlation dimension that steers the next pass. We show that this algorithm achieves good parallel scale-out, allowing it e.g. to generate *1.2TB* of correlated graph data in *h*alf an hour on a Hadoop cluster of *16 nodes*.

Contributions of our work are the following: (1) we propose a novel framework for specifying the generation of correlated graphs, (2) we show the usefulness of this framework in its ability to specify the generation of a social network with certain plausible correlations between values and structure, and (3) we devise a scalable algorithm that implements this generator as a series of MapReduce tasks, and verify the scalability of this data generator. In our vision, this data generator is a key ingredient for a new benchmark of complex graph analysis.

Outline. In Section 2, we present our framework for specifying the generation of correlated graphs based on correlation dimensions and correlation rules, and and describe how such a specification maps on a MapReduce-based algorithm. In Section 3 we apply our framework to the problem of generating a synthetic social network graph. The quality of the resulting social graph are evaluated in Section 4, and the scalability of our data generator is evaluated in Section 5. Finally, in Section 6, we review related work before concluding in Section 7.

2 Scalable Structure-correlated Social Graph Generator (S3G2)

In order to simulate the correlations in graph data, we propose a novel framework for Scalable Structure-correlated Social Graph Generator (S3G2) targeting at modeling huge correlated directed labeled graphs. In this section, we first formally define the end product of S3G2 which is a graph model of correlated data. Then we describe main building blocks (ingredients) for the generator, and the way they are combined in order to generate the graph (algorithm).

2.1 The Graph Model

We call the end product of the S3G2 graph model a *correlated property graph*.

Definition 1. *A S3G2 correlated property graph is a graph G(V, E, P, C) where V is a set of nodes, E is a set of edges, P is a set of properties and C is a set of classes.*

$$V = \bigcup_{r \in C} O_r \cup L$$

$$P = \left\{ P_{L(x)} \,|\forall x \in C \right\} \cup \left\{ P_{R(x,y)} \,|\forall x, y \in C \right\}$$

$$E = \left\{ \begin{array}{l} (n_1, n_2, p) | \left\{ n_1 \in O_x \wedge n_2 \in L \wedge p \in P_{L(x)} \right\} \\ \cup \left\{ n_1 \in O_x \wedge n_2 \in O_y \wedge p \in P_{R(x,y)} \right\} \end{array} \right\}$$

in which O_r is an object of a class r in C; L is the set of literals; $P_{L(x)}$ is set of literal properties; $P_{R(x,y)}$ is the set of relation properties.

In S3G2 graph, a node belongs to one of the object classes or is a literal. A labeled edge contains two nodes and an edge property in which one node belongs to an object class and the other node is a literal or an object. The edge property is then a literal property or a relation property, respectively. One node can have many edges with the same edge property and there is no edge connecting two literal nodes. In an edge, the end node is considered as the *property value* of the start node.

Figure 1 shows an example of S3G2 graph. In this example, the set of nodes V = {$P1$, $P2$, $P3$, $P4$, Student, Person, "Sophie", "Amsterdam", "Uva"} in which the last three nodes are literal nodes and the other nodes belong to some object classes. The set of properties P contains relation properties such as $< is >$, $< belongTo >$, $< knows >$ and literal properties such as $< name >$, $< liveAt >$, $< studyAt >$. Accordingly, there are edges between object nodes whose labels are those of the relation properties (e.g., ($< Student >$, $< belongTo >$, $< Person >$)), and edges between an object node and a literal node whose labels are those of literal properties (e.g., ($P1$, $< studyAt >$, "Uva")). Note that, hereafter, we also use this example for other descriptions.

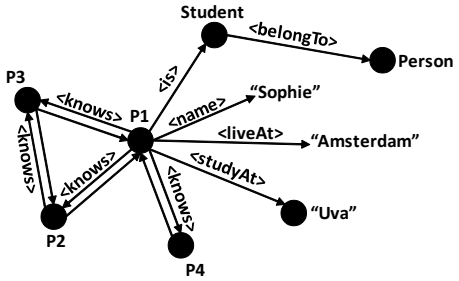


Fig. 1. Example of a S3G2 graph.

Correlation dimensions

Aiming at modeling correlated data in the graph, various types of data correlations are addressed in S3G2, including the correlations between property values (e.g., the value of property $< name >$ is correlated with values of property $< liveAt >$), the correlations between property values and graph structures (e.g., the availability of edges labeled $< knows >$ between nodes is correlated with values of properties $< studyAt >$, $< liveAt >$), and the correlations between the graph structures themselves (e.g., the existing of edge ($< P2 >$,$< knows >$,$< P3 >$) is correlated with the existing of edges ($< P1 >$,$< knows >$,$< P2 >$), ($< P1 >$,$< knows >$,$< P3 >$)). In order to generate the correlated graph data, our proposed framework is built on the notions of

- *correlation dimensions* that represent the underlying causes of (some) correlations. The correlation dimension can simply be a property value, e.g., the location name for the property $< liveAt >$.
- *dictionaries* with a *parameterized frequency distribution* that is used for generating the property values.

Correlation dimensions influences (i) property values of node by changing the frequency distributions of the value in dictionaries that generate these property values, and (ii) the connectedness, i.e., edges between pairs of two nodes in the graph. Assuming that we can enumerate a

correlation dimension, the frequency distributions are functions of these correlation dimensions. The underlying idea is that nodes which are close in one dimension of the dimensional space have a higher probability to be connected by certain types of edges.

Correlation dimensions are thought of as uni-dimensional domains; even in case where they are in reality multi-dimensional (such as in case of geographical information). We propose the use of space-filling curves to reduce multi-dimensional dimensions to a single dimensional domain.

Correlation dimensions may influence each other, i.e. it may be the case that they are not fully orthogonal. For instance, location, age and gender may well influence interests. Hence correlation dimensions can either be independent or depend on some other correlation dimensions.

In the graph generation process, property values will be created, influenced by these correlation dimensions. This we call *direct value correlation*. Subsequently, the property values themselves may influence other properties, or the structure of the graph (e.g., edges). This we may call *indirect correlation*; as a first correlated randomized process has determined certain properties; and a subsequent correlated randomized process then determines from that e.g., friendship structure.

As an example, the university one visits would be correlated to location and age; a direct value correlation. Then university, age, and gender might determine interests, which is a mix of direct (age, gender) and indirect (university) factors. Finally, commonalities between two people in university, age and interest may strongly influence the friendship structure. This is an indirect structural correlation.

Correlation rules

In S3G2, a correlation rule is of the form $A \mapsto (B, f(A))$, meaning that the values assigned to B are distributed according to a distribution function f built on the set of the values of the correlations A. Depending on whether A or B is a correlation dimension, a property value or a graph structure (e.g., edges between nodes) , these rules can be classified into three main categories:

1. The “correlation dimensions - property values” correlation rules

$$CR_{CP} = \{\, A \mapsto (B, f(A)) |\, A \in S_c \wedge B \in S_c \cup P\}$$

2. The “correlation dimensions - graph structure” correlation rules

$$CR_{CE} = \{\, A \mapsto (B, f(A)) |\, (A \in S_c \cup P) \wedge B \in E\}$$

3. The “graph structure - graph structure” correlation rules

$$CR_{EE} = \{\, A \mapsto (B, f(A)) |\, A \in E \wedge B \in E\}$$

where S_c is the set of correlation dimensions.

CR_{CP} contains rules for distributing the property value of each node given the values of correlation dimensions. Note that as the correlation dimension can be a property, these rules include correlation rules between property values. For example, considering the generation of a graph with the personal name and the geography information as property values of each node, one of the correlation rule in CR_{CP} can be ”location $l \mapsto$ (personal names, exponential distribution function $f(l)$)”. That is, the personal names distribute exponentially according to each location value.

CR_{CE} contains the rules between the correlation dimensions and the graph structure. The existing of edges between pairs of nodes depends on the values of each node. Specifically, the probability that an edge (u, v, p) is created is influenced by the property values of nodes u, v.

CR_{EE} contains the correlation rules between the graph structures which specify the probability of creating a graph structure in considering generated graph structures. Specifically, the

probability that an edge between two node u and v is created depends on the existing edges of nodes u and v.

These correlation rules should form a directed graph, called *correlation rules graph*, in which a directed edge from a node u to a node v represents a correlation rule from $u \mapsto v$. A node in this graph represents for a correlation dimension, a property or a graph structure. Figure 2 shows an example of a correlation rules graph. In this example, CR_{CP} = { $n_1 \mapsto n_2$, $n_1 \mapsto n_3$, $n_1 \mapsto n_4$, $n_4 \mapsto n_3$, $n_4 \mapsto n_7$ }, CR_{CE} = { $n_1 \mapsto n_5$, $n_2 \mapsto n_5$, $n_3 \mapsto n_5$, $n_4 \mapsto n_6$}, and CR_{CP} = { $n_6 \mapsto n_8$, $n_6 \mapsto n_9$, $n_7 \mapsto n_9$ }.

In S3G2, we assume that any subgraph (i.e., graph formed by a set of node and all edges between them) of this graph has nodes that do not have any incoming edges. The underlying idea of this assumption is that the generator can recursively follow these nodes for handling correlation rules. Specifically, S3G2 first generates the values for the nodes of the original graph, remove these nodes, and then recursively generate the data for the starting nodes of the new graph considering the data of removed nodes. Commonly, the correlation dimensions are the starting nodes of the whole correlated rule graph. Considering any arbitrary subgraph such as the subgraph of nodes n_4, n_6, n_7, n_8, there exists at least one starting node, e.g., n_4. For this correlation rule graph, to conform all the correlation rule, S3G2 can create the data in the order: n_1, n_4, n_2, n_3, n_7, n_5, n_6, n_8, n_9.

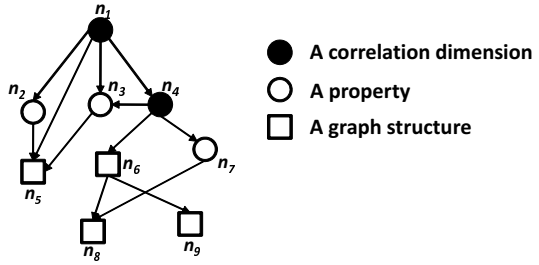


Fig. 2. Example graph of correlation rules.

2.2 Ingredients: Dictionaries and Frequency Distributions

The ingredients for S3G2 framework are the dictionaries of real property values and their frequency distributions. Each dictionary contains a collection of real data values for a particular property such as the geographical information, user's names, etc. In a dictionary, property values are distributed according to the frequency distribution function associated with that dictionary. The frequency distribution function is built based on the correlation between the property values and the correlation dimensions.

Definition 2. *Dictionary $D(A, C_1, .., C_k, f_D)$ of a property A contains a collection all values of that property; collections C_i (i=1,..,k) of k dimensions that are correlated with the distribution of property A; and a function $f_D(pv, c_1, .., c_k) \rightarrow R$ that returns the frequency distribution for each value pv of the property A in correlation with the values $c_1, ..., c_k$ of correlation dimensions $C_1, .., C_k$.*

In S3G2, for each dictionary, by using the frequency distribution function, the cumulative distribution value of each property value can be computed per set of correlation dimension values.

Then, for a random input value r, and a set of correlation values (c_1, c_2, ..., c_k), the dictionary returns a property value p that its cumulative distribution value is the smallest cumulative value greater than r. Figure 3 shows an example of a property dictionary. In this example, for the correlation values (c_1, c_2), if the input random value r is 0.2, the property value returned by the dictionary is pv_3 as the cumulative distribution value of pv_3 0.30 is the smallest value greater than r.

Correlation value	Property value	Cummulative Distr.
(C_1, C_2)	pv_1	0.10
	pv_2	0.15
	pv_3	0.30
	...	...
	pv_h	1.00
(C'_1, C'_2)	pv_1	0.20
	...	...
...	...	...

Fig. 3. Example of a property dictionary.

Frequency distribution function

We are not aiming at proposing a fixed frequency distribution function for all dictionaries since depending on the characteristic of each property, the real property values may distribute differently. In S3G2 we emphasize on the data correlation and build the frequency distribution function on the correlation dimensions.

For making frequency distributions a function of the correlation dimensions, S3G2 first ranks all the values of the property according to dimension values correlated with that property. The ranking reflects the popularity of a property value in considering all the correlated dimensions. Then, it fed these ranks into a fixed distribution (e.g., normalized distribution). As an attempt to simulate most realistic data distribution, in S3G2, we use geometric distribution — a discrete analog of the exponential distribution — for the fixed distribution. Specifically, a generic frequency distribution function f_D for a property value pv is formulated as follows.

$$f_D(pv, c_1, .., c_k) = (1 - p_D)^{k-1} \times p_D \tag{1}$$

where k is the rank of pv among all the values of property A sorted according to their popularity in considering dimension values $c_1, .., c_k$. Parameter $0 < p_D < 1$ can be flexibly selected according to each dictionary D.

Theorem 1 can be used for randomly get a value from the dictionary. This theorem can be proved by referring to the geometric method for generating the next edges in [3].

Theorem 1. *For a dictionary in which the data values have the frequency distribution of the Fomular 1, with a random value r, the value that is returned from this dictionary is the value at rank k where*

$$k = 1 + \left\lfloor \log(1 - r) / \log(1 - p_D) \right\rfloor \tag{2}$$

2.3 Generating Algorithm

The generator takes the correlation rule graph as an input of the generating algorithm. It first parses this graph to identify the order of data to be generated, e.g., which correlation dimensions or properties are generated first, which properties or structures will be created next, etc. As explained in Section 2.1 for the rule graph, S3G2 uses the strategy of recursively following starting nodes to find the generating order.

Generally, S3G2 breaks the whole generating process into three main steps: 1) Generate the "properties space" in which all the correlation dimensions (the source of the data correlations) and properties of each node in the correlated graph are generated. 2) Generate the graph structure (i.e., edges between each pair of two nodes) that is directly correlated with the correlation dimensions and the property values; 3) Generate the graph structures that are indirectly correlated with the correlation dimensions. In following, we describe these steps in detail and use the Figure 2 as the example.

Step 1. Generating properties space

In this step, the generator creates the values of the correlation dimensions and the properties while considering correlation rules in CR_{CP}. It is done by following the starting nodes in the correlation rule graph. Generally, as correlation dimension can be inter-dependent, values of correlation dimensions that are independent from others are generated first, then the values of remaining correlation dimensions are created. After that, the values of properties are generated from their dictionaries value.

As the example in Figure 2, values for correlation dimension n_1 will be generated first, then the values of n_4. Subsequently, the values of n_2, n_3, n_7 are generated from the dictionary of these properties built on correlation dimensions n_1, n_4.

Step 2. Generating direct correlated graph structure

This step generates the graph structure (e.g., edges) while considering correlation rules in CR_{CE}. In the example in Figure 2, the structures n_5, n_6 and n_7 are created in this step. The creation of these structure are influenced by the values of the correlation dimensions and properties in each node, e.g., creation of n_5 is influenced by n_2, n_1 and n_3.

For creating the graph structure, existing random graph generators typically need to keep large data structures in memory, at least linear to the size of the graph in terms of nodes and edges. This is unacceptable for S3G2. In order to address this problem, we propose a novel approach using a window sliding along all the nodes of the graph. Specifically, S3G2 divides the nodes and the edges in a huge graph into so-called tiles. Each tile holds all properties of the node as well the edges from these nodes. In S3G2, as a principle, nodes in a tile are close to each other in a correlation dimension, i.e., the dimension values assigned to these nodes are similar. The idea is that nodes having similar values of correlation dimension and properties tend to be connected. As a real example, people studying in the same university and having similar interests are commonly connected by a friendship.

Definition 3. *A tile is a set of k graph nodes with their properties and edges information in which k is a predefined number.*

The generator then creates a memory-resident window containing a specific number of tiles so that the size of the window is fitted in the main memory. The window then slides along the whole graph. For each sliding time, a subset data for all the nodes in one tile is generated. Then, the tile with generated data will be removed and a new tile will be appended into the window. This process keeps running until the data for all nodes is generated. An example in Figure 4 intuitively shows the basic idea of this approach.

Multi-passes over correlation dimensions

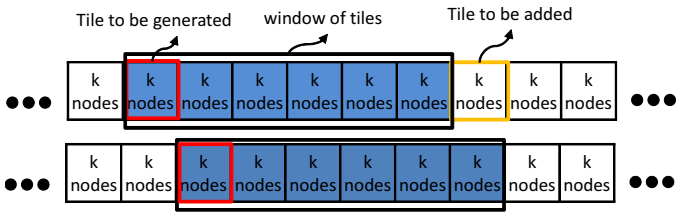


Fig. 4. Sliding window along the graph.

The correlation dimension are the source of creating correlated data including the property values and the graph structure. As there are many correlation dimensions, in order to create the edges in the correlated graph considering data correlation, S3G2 makes multiple sequential passes over these tiles, keeping a window of tiles in memory only. In each pass, a part of data that is correlated with a correlation dimension is generated. We offer the "*sliding speed*" of the window in each pass as a parameter for indicating the proportion of the edges that will be generated in each pass.

Definition 4. *Sliding speed of the window along a pass is the inverse of the portion of edges that need to be created for all the nodes in one tile before the window slides to the next tile.*

As an example, assuming that window slides in three passes and the portions of the edges that need to be generated in each pass are 0.2, 0.3, and 0.5. Then, the sliding speeds of the window in these passes are 5 ($=\frac{1}{0.2}$), 3.33 ($=\frac{1}{0.3}$), 2 ($=\frac{1}{0.5}$), respectively. Intuitively, window slides faster if a smaller number of edges needs to be generated for a tile of nodes. Thus, the higher the value of sliding speed is, the faster the window slide in a pass.

If there are m correlation dimensions: D_1, D_2, D_3, ., D_m The sliding speeds of the window in these dimensions are s_1, s_2, ., s_m where $\sum_{i=1}^{m} s_i = 1$. Then, for a node with degree of h, the number of edges created for that node in each pass are $h \times s_1$, $h \times s_2$, ., $h \times s_m$, respectively.

Each dimension D_i has a distribution function FD_i for mapping the value of that correlation dimension to a point in the dimensional space. Two similar values should be mapped to two points close to each other in the dimension co-ordinate. Figure 5 shows an example of the three dimensional space with two nodes and their distribution value in each dimension co-ordinate.

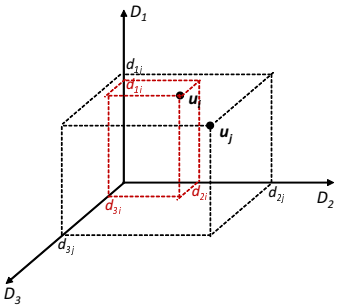


Fig. 5. Example of social space with 3-dimensions.

Algorithm 1 GenerateEdges(t, α, β, pos)

Input: t: dimension t that the window is sliding along
Input: α: base probability for creating edge
Input: β: exponential rate
Input: pos: Current position of the window (number of traversed tiles)

```
1: Append new tile of nodes into window.
2: for each node u in first tile do
3:    for each node v in the window do
4:       Generate a uniform random number r in [0,1)
5:       Connecting probability prob = α · β^(d_tu − d_tv)
6:       if r < prob then
7:          createEdge(u,v)
8:       end if
9:       if reachRequiredEdgeNumber(u) then
10:         continue
11:      end if
12:   end for
13: end for
14: GenerateEdges (t, α, β, pos + 1) //window slides
```

Under the underlying assumption that nodes tend to interact with others close on at least one of the correlation dimensions, we only generate edges (e.g., user friendship) for nodes that are close to each other in a correlation dimension.

In each of the m passes, we will generate data correlated with one correlation dimension; and during this pass, all nodes which are close to each other will at some point in time have resided in RAM together; as "being close" equates at a tile distance less than the window size.

Thus, for each tile of nodes, we can generate all the edges correlated with a dimension by considering tile neighbors in the sliding window. Note that two nodes which are close in a dimension may be far in the other dimensions and do not belong to a sliding window in those dimensions.

While the window slides along all the nodes in a dimension t, the probability that an edge is created between two nodes based on their distances in this dimension.

$$\Pr obE_t(i,j) = (1-\gamma_t)^{d_{ti}-d_{tj}} \times \gamma_t \tag{3}$$

γ_t: the base probability of creating an edge along dimension t

The steps of creating edges while sliding the window along a dimension t is shown in Algorithm 1.

Each FD_i function for a dimension D_i is built based on the real data value of that correlation dimension. If the data value of that correlation dimension is multi-dimensional itself (e.g., a value of the location dimension contains longitude and latitude information which is two-dimensional), building a function that maps similar data values to close points in the correlation dimension is not trivial. However, if the data value is a one-dimensional value that can be easily ordered (e.g., a value of age dimension), the FD_i can simply be a bijection function from the data value to it ranks in the ordered set of all the values.

Since $z-order$ space-filling curve can map multidimensional data to one dimension while preserving the locality of the data points, in S3G2, we use $z-order$ for those correlation dimensions that the values are represented in multi-dimensions. Specifically, for each dimension D_i, FD_i is the rank of the data value in the $z-order$.

Step 3. Generating indirect correlated graph structure

This step generates graph structures that are indirectly influenced by the correlation dimensions. "Indirectly" here means that the creation of these graph structures, e.g., edges between each pair of two nodes u, v, depends on the current generated edges of nodes u, v and the properties of these nodes. It handles the correlation rules in CP_{EE} as the to-be-created structures are influenced by existing generated graph structure. In example in Figure 2, the structures n_8 and n_9 are generated in this step.

Similar to Step 2, the window slides along all nodes in order to generate the edges for each node. Note that the window does not need to do multi-passes sliding along all correlation dimensions in order to create edges indirectly correlated with these dimensions. The data in each tile contains all the properties and generated edges of nodes that influence the creation of new edges. The probability that a new edge is created between two nodes in this step depend on the rules defined in CP_{EE}.

3 A case study: Modeling a social network

In this section, we demonstrate the use of S3G2 framework by practising a use case of modeling a social network. A social network (SN) can be simply viewed as a graph, namely, a social graph, in which each node represents a user and an edge between two nodes represents the friendship between two corresponding users. In this case study, the output graph $G(V, E, P, C)$ contains users and entities of social activities (e.g., discussions) as the object classes of C. These object classes and their attributes value (e.g., user's name, post's created date, ...) form the set of nodes V. E contains all the connectivities between two users including their friendship edges and social activity edges between users and a social activity when they all join a social activity (e.g., users discuss about one topic). Note that there is no edge between any two social activities. P contains all attributes of a user profile, the properties of user friendships and social activities.

We consider three correlation dimensions in modeling a social graph including the location, the interest, and a random dimension. We will discuss the rules in CR_{CP}, CR_{CE} and CR_{EE} during the description of building dictionaries.

3.1 Dictionaries for a social network

As the ingredients of the social graph generator, a number of dictionaries of real data for generating the properties in the social network have been used. Most of the dictionaries such as the dictionary of personal names, dictionary of popular places, etc. are collected from the linked open datasets (e.g., DBPedia). Detailed description for collecting these dictionaries can be found in wiki page of Social Intelligent benchmark[3]. In this section, we represent the building for several dictionaries, their correlation dimensions, and distribution functions.

In a social network, the generic frequency distribution function 1 is applied for a number of dictionaries where the values of each property can be realistically ordered according to their popularies while considering the correlation dimensions. For example, the dictionary of surnames/givennames, the dictionary of interests (hobbies), the dictionary of internet browsers, the dictionary of internet browsers,...

Considering the dictionary of given names, dictionary of surnames, we observed that the frequency distributions of these properties are correlated with the location dimension. Thus, these dictionaries are built with considering the location dimension. Specifically, the data values of a property are ranked according to their real popularities per location. Then, the frequency of each

[3] http://www.w3.org/wiki/Social_Network_Intelligence_BenchMark

data value is computed by using the Function 1 in which set of $C_1, C_2, ., C_k$ only consists of the location dimension. Table 1 shows the ranks of each value in the surname dictionary considering location dimension values Germany and Russia.

Table 1. Ranks of surnames sorted by their popularities according to the location dimension

Location = Germany		Location = Russia	
Rank	Surnames	Rank	Surnames
1	Muller	1	Smirnov
2	Schmidt	2	Ivanov
3	Schneider	3	Popov
...	...	...	...
135	Ivanov	112	Schneider
...	...	...	...

For dictionaries where the real popularities of the data values not clear, we use heuristics for finding their functions of frequency distribution. Following are examples of these dictionaries.

Dictionary of locations: The frequency of each location is generated proportionally according to the population of each location (e.g., a country). Specifically, frequency value of a location L in the dictionary is computed by using the following function.

$$f(L) = \frac{population\_of(L)}{total\_population} \tag{4}$$

3.2 Generating steps

For modeling the social network, we follow three steps of S3G2 algorithm.

Step 1. We generate all the properties of user profiles as well as the number of friends for each user. As most of the real life social network follow power-law distributions for the number of user's friends, we randomly generate power-law distributed for all nodes in the social graph. (i.e., the number of nodes of degree h is proportional to h^{λ}).

$$P(h) \sim c \times h^{-\lambda} \tag{5}$$

$P(h)$: probability that a node has degree h

c: normalization constant

λ: a parameter for the power-law distribution. In a social network, typically, $2 < \lambda < 3$

Step 2. As we mentioned, in our social graph, we only consider three correlation dimensions including location, interest, and random dimensions. This can be easily extended.

For the location dimension, since location is a multi-dimensinal data with the latitude and longitude values. We use the $z-order$ space filling curve in order to convert this multi-dimensional data to uni-dimensional data. 2, we use $z-order$ for those correlation dimensions that the values are represented in multi-dimensions. Specifically, for each dimension D_i, FD_i is the rank of the data value in the $z-order$. Following is the function for mapping the location dimension into its co-ordinate in the dimension space.

$$FD_{location}(U_j) = rank_j(z - value(long_j, lat_j)) \tag{6}$$

where $long_j$, lat_j are the longitude and the latitude of the location that user U_j is living

For the interest dimension, in SIB, we simplify this to represent popular singers as the set of user's interests. For mapping a user's interest into the co-ordinate of the interest dimension, we order the singers according to their debut year with the observation that singers debut in the same year may sing the similar music genre (e.g., the music genre popular in that year). People who like these singers tend to interact with each other.

$$FD_{interest}(U_j) = rank(debut\_year(s)) \tag{7}$$

where s is singer that user U_j likes.

For the random dimension, we simply generate a value in (0,n] where n is the total number of users by using a random generator with the seed value is the $userId$. The purpose of sliding the window along the random dimension is to shorten the average path length between each pair of nodes in the social graph.

$$FD_{random}(U_j) = n * rand(j) \tag{8}$$

where $rand(j)$ generates a random value in (0,1)

We slide the window for three passes along each of the correlation dimension and create the friendship edges for all the users. The probability 3 is applied for generating friendship between two users in a correlation dimension.

Step 3. In this step, we generate the social activities for all the users. Generally, in a social network, users directly connect to each other by friend relationships (generated in the Step 2). However, they are also indirectly connected to others by joining same social activities such as writing posts and comments in a discussion, upload a photo of them, etc.

4 Social Network Characteristics

Existing literature studying social networks have shown that popular real social networks have the characteristics of a small-world network[11, 14, 4]. Considering the three most robust measures, i.e. the social degrees, the clustering coefficient, and the average path length, of the network topology, a social network has the following characteristics:

- A power-law distribution of the social degrees: The number of friends for each user follows a power-law distribution.
- High clustering coefficient: The clustering coefficient is used to measure the local connectivity of a node (i.e., a user) in a social graph. It is computed by the density of an undirected graph formed by that user and her friends, in which the density of a graph $G(V, E)$ is $|E|/(V * (V - 1)/2)$ where V is set of vertices, E is set of edges.
  Wilson et al[14] have shown that in a social network, low-social-degree users tend to have high clustering coefficients, which indicates tight local connectivities at the edge of the social graph.
- Low average path length: The average path length of a social graph is the average of all-pairs shortest paths in the graph. Existing experimental results show an average path length of 6 or lower for a social network. This average path length can be proportional to the $log(n)$, where n is the total number of users in the social network.

We empirically show that the generated social network conforms to the "small-world" characteristics. In our experiment, we generated the social graphs of 10K, 20K, 40K, 80K, and 160K users. Each tile contains 10 users, each sliding window has 20 tiles.

Table 2. Graph measurements of the generated social network.

# users	Diameter	Avg. Path Len.	Avg. Clust. Coef.
10000	5	3.13	0.224
20000	6	3.45	0.225
40000	6	3.77	0.225

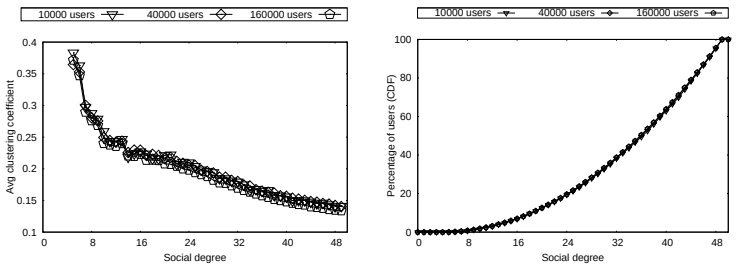
Clustering coefficient

Table 2 shows the graph measurements of the generated social network while varying the number of users. According to the experimental results, the generated social networks have high clustering coefficients of about 0.22 which adequately follow the analysis on real social networks in [14] where the clustering coefficients range from 0.13 to 0.21 for Facebook, and 0.171 for Orkut. As showed in Figure 6(a), the clustering coefficients distributions according to the social degrees indicate the small-world characteristic of tightly clustered fringe in social networks. That is, the low-degree nodes (i.e., users) have high clustering coefficients.

Average path length

Table 2 shows that the average path lengths of generated social graphs range from 3.13 to 3.77 which are comparable to the average path lengths of real social networks observed in [14]. These experimental results also conform to the aforementioned observations that the average path length can be proportional to the log of the total number of users in the social network. (Add text for the diameter and radius)

Since we used a simple all-pair-shortest-path algorithm which consumes too much memory for analyzing very large graph, Table 2 only shows the results of the average path length for a social graph of 40K users. We also note that the input parameters for S3G2, e.g., "the number of maximum friends per user" can be flexibly changed so that a social network with such a required characteristic, e.g., lower path length, is created. Due to the space limitation, we do not discuss in details about how to tune the S3G2 parameters.



(a) Clustering coefficient distributions (b) Cumulative user distributions by the social degrees

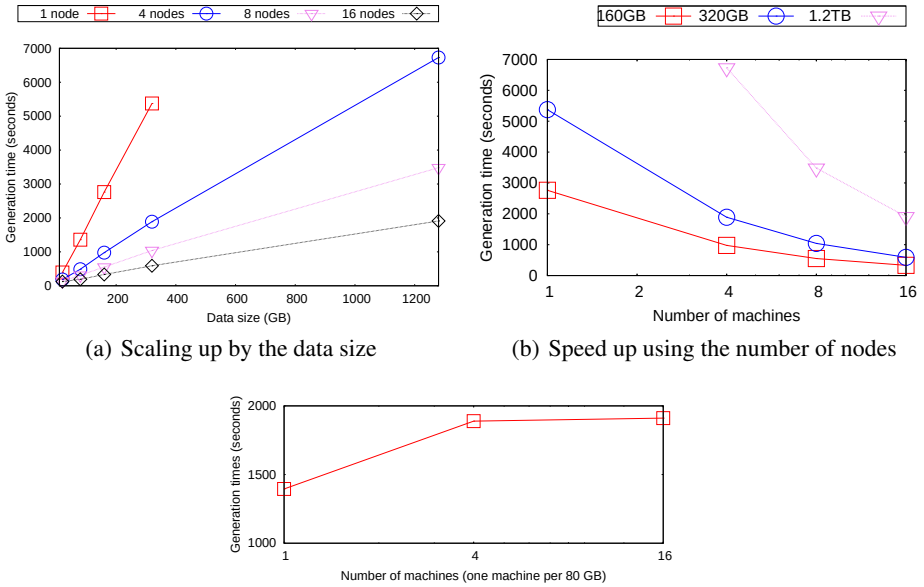
Fig. 6. Distributions of data in the generated graph

Social degree distributions

Figure 6(b) shows the cumulative distributions of the social degrees with different number of users. According to the experimental results, for all cases, the social degrees follow power-law distributions with the alpha value of about 2.0.

5 Scalability Evaluation

In order to show the scalability of S3G2, we conduct experiments on a clustering of 16 nodes which are connected by 1 Gigabit Internet. Each node is a PC with an Intel i7-2600K, 3.40GHz CPU, 4-core CPU and 16 GB RAM. We devise the generating algorithm on the clustering system by using Map-Reduce library of Hadoop[4]. Then, we generate the dataset by up to 1.2 TB which, to the best of our knowledge, is the biggest realistic structure-correlated social graph that has been generated so far.



(a) Scaling up by the data size

(b) Speed up using the number of nodes

(c) Scaling out

Fig. 7. Scalability of S3G2

In Figure 7(a), for a specific number of nodes, we increase the data size. Experimental result shows that the generation times are linear according to the size of the data, which approves that the data generator is scalable according to the size of the graph.

Figure 7(b) shows the speed up of the generator by adding more nodes for a particular data size. The trends of three lines corresponding to data sizes of 160GB, 320GB, and 1.2TB show that S3G2 speeds up faster for large data size.

Figure 7(c) shows the scalability of S3G2 by scaling out the system while fixing the number of nodes per a specific amount of data size, i.e., one node per 80 GB. In this figure, the generated data size in increased by factor of 4 (i.e., the data sizes are 80GB, 320GB, and 1.2TB, respectively), and correspondingly increasing number of nodes by factor of 4 (i.e., single node, 4 nodes, and 16 nodes). The experimental result show that it is almost linear when scaling out from 4 machines

[4] hadoop.apache.org

to 16 nodes. Thus, it is convinced that S3G2 can generate data of any size in a reasonable amount of time by increasing the nunber of nodes. Here, the dataset of 1.2 TB data can be generated in about half hour.

6 Related Work

There are number of works studying the characteristics of social network[10, 6, 11, 14, 4, 1, 8] and the generator of random graph having properties of a social network [13, 2, 3, 9, 5, 7]. However, to the best of our knowledge, there is no generator that creates a synthetic social graph with realistic data correlations. The existing graph generators mostly consider the topology and the structures of the generated graph.

One of the first work to generate social-network-like random graph is done by Watts et al.[13]. In this generator, the authors created a random graph having such properties of small world network as high clustering coefficient and low path lengths by connecting a node with k nearest neighbors and then rewiring edges. To satisfy the degree distributions, Barabasi et al.[2] introduced the model of preferential attachment which was subsequently improved by Batagelj et al[3]. The main idea of this model is that, for a new vertex, the probability that an edge is created between this vertex to an existing vertex depending on the degree of that vertex. Leskovec et al.[9] proposed a tractable graph that matches several properties of a social graph such as small diameter, heavy-tails in/out degree distribution, heavy-tails eigenvalues and eigenvectors by recursively creating a self-similar graph based on Kronecker[5] multiplication. None of these algorithms considers the correlation of a user's attributes in the social graph.

Recently, Bonato et al.[5] studied the link structure of social network and provided a model that can generate a graph satisfying many social graph's properties by considering the location of each graph node in geometric and ranking of each node. In this model, each node is randomly assigned a unique rank value and has a region of influence according to its rank. The probability that an edge is created between a new node and an existing node depends on the ranking of the existing node. Similar to the approach of using influent regions, Foudalis et al.[7] practiced social users' behavior and constructed a set of cliques (i.e., groups) over all the users. For each new node (i.e., a new user), an edge to an existing node is created based on the size of cliques they have in common. These models are reaching the realistic observation that users tend to join and connect with people in a group of same properties such as the same location. However, the simulation of realistic data correlations is quite limited and both do not address the correlations between different attributes of the users.

Besides, all of the existing models need a large amount of memory for storing either the whole social graph or its adjacent matrix. Leskovec et al. [9] may need to store all stages of their recursive graph. Although Batagelj et al. aimed at providing a efficient space-requirement algorithm, the space-requirement is $O(|V| + |E|)$[3] where V is the set of vertices and E is the set of edges. This is not feasible when we need to generate a huge social graph of million nodes where each node contains many properties. Thus, a solution that only a part of graph needs to be stored in the main memory is required.

7 Conclusion

In this paper, we have proposed a novel framework for scalable graph generator that can generate huge graphs having correlations between the graph structures and the graph data such as node

[5] http://en.wikipedia.org/wiki/Kronecker_product

properties. While current approaches at generating graph require holding it in RAM which is not sufficient for a huge graph, our graph generator (S3G2) can generate the graph with little memory and in parallel by using a sliding window protocol. In order to address the problem of generating correlated data and structure together, which has not been handled in existing generators, we propose an approach that slides the window for multiple passes where each pass visits data along a correlation dimension.

We demonstrate the facility of our framework by applying it to the scenario of modeling a social network graph. The experiments show that our generator can easily generate a graph having all the important characteristics of a social network while considering the correlations in the graph data. Besides, the implementation on Map-Reduce framework not only shows the scalability of our generator but also convince that the generating algorithm can be easily devised to perform on a clustering system.

References

1. Y. Ahn, S. Han, H. Kwak, S. Moon, and H. Jeong. Analysis of topological characteristics of huge online social networking services. In *Proceedings of the 16th international conference on World Wide Web*, pages 835–844. ACM, 2007.
2. A. Barabási, R. Albert, and H. Jeong. Scale-free characteristics of random networks: the topology of the world-wide web. *Physica A: Statistical Mechanics and its Applications*, 281(1-4):69–77, 2000.
3. V. Batagelj and U. Brandes. Efficient generation of large random networks. *Physical Review E*, 71(3):036113, 2005.
4. F. Benevenuto, T. Rodrigues, M. Cha, and V. Almeida. Characterizing user behavior in online social networks. In *Proceedings of the 9th ACM SIGCOMM conference on Internet measurement conference*, pages 49–62. ACM, 2009.
5. A. Bonato, J. Janssen, and P. Prałat. A geometric model for on-line social networks. In *Proceedings of the 3rd conference on Online social networks*, pages 7–7, 2010.
6. I. de Sola Pool and M. Kochen. Contacts and influence. 1978.
7. I. Foudalis, K. Jain, C. Papadimitriou, and M. Sideri. Modeling social networks through user background and behavior. *Algorithms and Models for the Web Graph*, pages 85–102, 2011.
8. H. Kwak, C. Lee, H. Park, and S. Moon. What is twitter, a social network or a news media? In *Proceedings of the 19th international conference on World wide web*, pages 591–600. ACM, 2010.
9. J. Leskovec, D. Chakrabarti, J. Kleinberg, and C. Faloutsos. Realistic, mathematically tractable graph generation and evolution, using kronecker multiplication. *Knowledge Discovery in Databases: PKDD 2005*, pages 133–145, 2005.
10. S. Milgram. The small world problem. *Psychology today*, 2(1):60–67, 1967.
11. A. Mislove, M. Marcon, K. Gummadi, P. Druschel, and B. Bhattacharjee. Measurement and analysis of online social networks. In *Proceedings of the 7th ACM SIGCOMM conference on Internet measurement*, pages 29–42, 2007.
12. M. Stillger, G. Lohman, V. Markl, and M. Kandil. Leo-db2's learning optimizer. In *Proceedings of the International Conference on Very Large Data Bases*, pages 19–28, 2001.
13. D. Watts and S. Strogatz. Collective dynamics of "small-world" networks. *Nature*, 393(6684):440–442, 1998.
14. C. Wilson, B. Boe, A. Sala, K. Puttaswamy, and B. Zhao. User interactions in social networks and their implications. In *Proceedings of the 4th ACM European conference on Computer systems*, pages 205–218. Acm, 2009.