An approach for generating random temporal semantic graphs with embedded patterns

Aurélie Leborgne, Jorick Nuss, Florence Le Ber, and Stella Marc-Zwecker

Université de Strasbourg, CNRS, ENGEES, ICube UMR 7537,
F-67000 Strasbourg, France
{aurelie.leborgne,stella}@unistra.fr, florence.leber@engees.unistra.fr

Abstract. Nowadays, many information can be represented by graphs with both semantic and temporal relationships. In this article, we formalize these graphs as temporal semantic graphs, called TS-graphs. Our goal is to randomly generate such graphs in order to use them as a test base to compare and assess graph mining algorithms. Our method makes it possible to generate random graphs by mastering the incrustation of predefined patterns. Indeed, our proposition is to build random graphs from randomly modified predefined patterns, random vertices and random edges. Finally, we provide a measure for assessing how the patterns are embedded in the rest of the graph, that could show how it will be difficult to extract them by usual graph pattern mining methods.

Keywords: Random graph generation, temporal and semantic graphs, simulation, graph patterns

1 Introduction

Today, many information can be represented by graphs, e.g. active connexions within neurons in a brain [26], pathways for animals [25], sampling sites on river networks [20], or the evolution of agrosystems [14]; these graphs often represent dynamic phenomena, and thus combine both semantic and temporal aspects. We call them temporal semantic graphs (TS-graphs) in the following. An example of such a graph is the model of spatio-temporal graph defined by Del Mondo *et al.* [11]. The size of graphs based on real data can be important, e.g. about 100 images of brain activity for an observation in functional magnetic resonance imaging, and thus need to be summarized to facilitate the analysis. A way to perform such a summary is to apply graph mining methods, and especially graph frequent pattern mining. Besides, to test and improve such methods, large sets of graphs are necessary. In order to complete existing repositories of labelised graphs based on real data (real graphs in the following), synthetic graphs can be used, but they need to have characteristics close to these real graphs. Furthermore, to represent the variety of real graphs, synthetic graphs should be random, with respect to some given constraints.

Besides, random generation of graphs is essential to design and test methods of graph frequent pattern mining [4]. Indeed, it is quite relevant to have a tool

to generate an infinity of graphs extremely varied in terms of size, structure and complexity, and especially fully configurable. We do not know in advance the difficulty of designing temporal semantic graphs mining methods, that can be applied in several domains; it is therefore interesting to prepare powerful tools to provide these tasks.

We are thus interested in the generation of random TS-graphs. Our major aim is to produce graphs with inexact recurring patterns "embedded" in a uniform stochastic generation of vertices and edges. The recurring patterns are produced by applying random transformations on a set of temporal semantic sub-graphs that we will call "source patterns". Thus we obtain a number of "transformed patterns", similar to the source patterns, that will be randomly inserted in the graph. Figure 1 illustrates these various points. Finally, the proposed method would be able to produce completely synthetic TS-graphs approaching graphs of real data in various application domains.

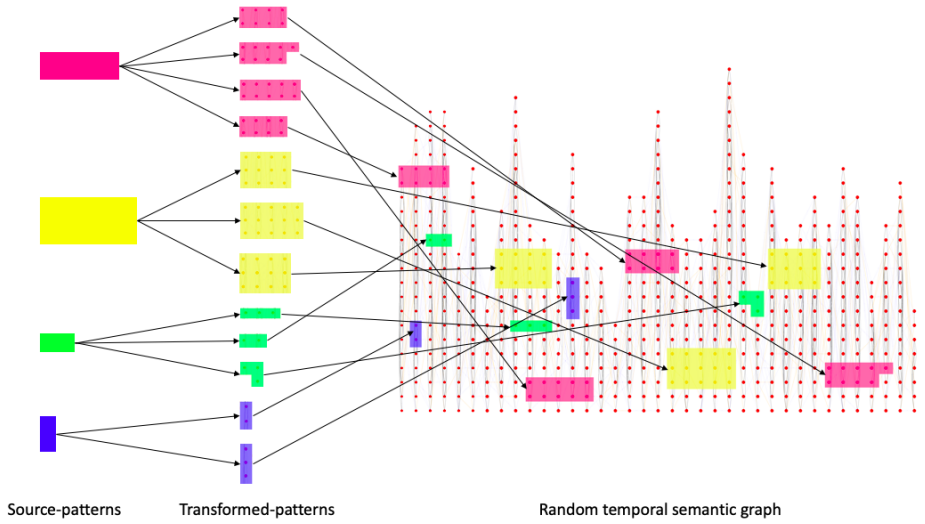


Fig. 1. Illustration of the generation method of random *TS-graphs* and its vocabulary: source-patterns (left) are randomly transformed (center) and then embedded in a temporal semantic graph (right)

The remainder of this paper is organized as follows. Section 2 is an overview on graph generation approaches while Section 3 describes the model of graphs we deal with. The generation method of random graphs is provided in detail in Section 4. Then, a measure for assessing pattern insertion into these generated graphs is presented in Section 5. Finally, Section 6 presents the conclusion of this paper with directions for future works.

2 Related Work

Generation of graphs allows to build test batteries at the level of performance, to evaluate effectiveness and efficiency of algorithms [2, 17], to simulate processes and data [15], to visualize and summarize [18] but also to compute statistics, *etc.* Moreover, graphs, and consequently simulated graphs, are widely used in many domains like environment, health, social networks, biology, web, business intelligence, *etc.*

In most cases, the goal is to generate realistic graphs. To do that, many models are proposed. The oldest classical models [12, 3], used for a long time, allowing to generate random graphs in a simple manner. The probability to generate an edge between two vertices is evenly equal to $\frac{|E|}{|V|^2}$, where $|E|$ is the number of edges and $|V|$, the number of vertices. These models generate a random graph that has a binomial degree distribution. They are adapted to certain situations, but cannot reflect the various properties of a real-world complex network, for example the Internet, the World Wide Web, social networks.

To generate this kind of complex graphs, adapted for natural or human systems, Barabási-Albert model [6, 5] is one of the most referred models. It consists in an algorithm, which uses a preferred attachment mechanism. In these graphs, some nodes have an unusually high degree compared to other nodes of the network, due to power-law degree distribution. Hadian *et al.* [13] worked on the acceleration of this model. Another well-known model for this kind of graphs is the Chung-Lu model [1], in which the probability of an edge is proportional to the product of the degrees of its two vertices. The problem of this model is that it requires a set of parameters, making it more difficult to use.

Another way to obtain synthetic graphs close to the reality is to use Kronecker graph models [18, 19]. Indeed, using the Kronecker's product, that is a non-standard matrix operation, allows to respect some properties of real graphs, e.g. heavy tails for the in- and out-degree distribution, heavy tails for the eigenvalues and eigenvectors, or small diameters.

To generate synthetic graphs, recursive models are also popular. The Recursive MATrix method (R-MAT) [7] is a simple and fast algorithm. It recursively selects a quadrant on adjacency matrix to generate an edge in dynamic manner, then the same procedure is repeated until obtaining the desired number of edges. Besides, Park and Kim [21] worked to get a fast and scalable generator (TrillionG) to generate trillion-scale synthetic graphs. To do that, authors based their recursive vector model on the generalization of Chakrabarti *et al.*'s model [7] and Leskovec *et al.*'s model [18, 19]. The RDyn approach was recently designed [24] for generating temporal graphs representing the dynamics of network communities. This method can fit our problem in some cases, since the dynamics of such communities involve splits and merges, as in the application domains we focus on.

In a more general context Kuramochi and Karypis [17] proposed an algorithm to generate random general graphs including known patterns to create synthetic datasets. This work was carried out with the aim of serving as a test base for

frequent graph pattern mining methods. Indeed, as patterns have been voluntarily included in the random graph, it is possible to test these methods, knowing the expected result. To do this, authors generally rely on a Poisson distribution. This is what we propose to do for the generation of temporal semantic graphs.

3 The model of TS-graphs

In the following, we introduce a general model for TS-graphs. We consider a set of temporal entities and their relations. Two types of relations are considered, general semantic relations (e.g. spatial, correlation, etc.) and filiation relations. Filiation relations are necessary to follow entities during time. The model is restricted such that a relation links two entities at the same time instance or at two consecutive time instances. Furthermore, there is at most one semantic relation and one filiation relation between two entities. The model can nevertheless be generalized for accepting several, but a limited number of semantic relations.

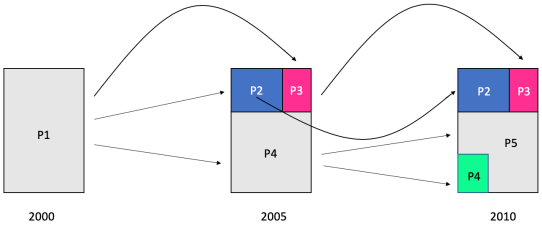
Based on the proposition of [10], TS-graphs are formalized as follows. We introduce a time domain, $\mathcal{T} = \{t_1, t_2, ...t_n\}$, where t_i represents a time instance of a given granularity and $t_i < t_{i+1}$ for all $i \in [1, n]$. Δ is a set of entities, $\{e_1, e_2, ...e_m\}$. We also introduce Σ, a set of semantic relations, and Φ, a set of filiation relations.

A temporal semantic graph $\mathcal{G}$ is a tuple (V, E_Σ, E_Φ), where V is a set of vertices $(e_l, t_i) \in \Delta \times \mathcal{T}$, E_Σ is a set of tuples $((e_i, t_i)T(e_j, t_j))$ where $(e_i, t_i), (e_j, t_j) \in V$, $t_i \leq t_j \leq t_{i+1}$, and $T \in \Sigma$, and E_Φ is a set of tuples $((e_i, t_i)\rho(e_j, t_{i+1}))$ where $(e_i, t_i), (e_j, t_{i+1}) \in V$, and $\rho \in \Phi$.

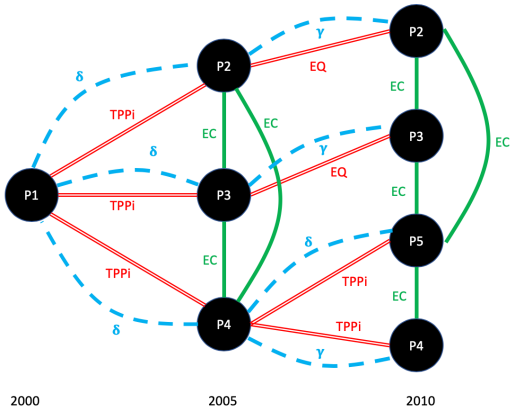
This model can be seen as the union of three subgraphs in which entities are grouped according to time instances. For example, Fig.2(a) represents the evolution of spatial regions at three time instances: $\mathcal{T} = \{t_{2000}, t_{2005}, t_{2010}\}$, $\Delta = \{P_1, P_2, P_3, P_4, P_5\}$ (P stands for *plot*). This evolution is represented within a semantic temporal graph $\mathcal{G}_1$ (Fig.2(b)), where semantic relations come from the Region Connection Calculus (RCC8) theory [23], to modelize topological inclusion and connection between spatial regions.

- The **subgraph of semantic relations** represents the semantic interactions between entities at a given time t_i. In Fig. 2(a), these relations (represented by green lines) are illustrated by the RCC8 relationship. For example, the nodes (P_2, t_{2005}) and (P_3, t_{2005}) of $\mathcal{G}_1$ are connected by an edge labelled with EC (Externally Connected).
- The **subgraph of temporal semantic relations** represents the semantic interactions between entities at two successive time instances. In the example of Fig. 2(a), these temporal semantic relations (red lines) are illustrated by the RCC8 relationship. For example, in $\mathcal{G}_1$, the nodes (P_1, t_{2000}) and (P_2, t_{2005}) are connected by an edge TPPi (Tangential Proper Part inverse), while (P_3, t_{2005}) and (P_3, t_{2010}) are connected by an edge EQ (EQual).
- The **subgraph of filiation relations** represents the transmission of identity between entities at different times (blue lines); following [11], two types of *filiation*

relations are considered, *continuation* and *derivation*, respectively denoted γ and δ in Fig. 2(a). A continuation relation between (e_i, t_i) and (e_j, t_{i+1}) means that e_i and e_j have the same identity; a derivation relation means that e_j contains a part of e_i identity (here a part of its area). For example, the nodes (P_4, t_{2005}) and (P_5, t_{2010}) are connected by an edge labelled with $\delta(derivation)$.



(a) Data on spatial regions over time



(b) The corresponding TS-graph $\mathcal{G}_1$

Fig. 2. A small example of TS-graph with spatial relations and entities

4 An algorithm for generating TS-graphs

Our aim is to build temporal synthetic graphs, that include some regularities or frequent patterns (i.e. TS-subgraphs that appear with a frequency higher than a user-specified or auto-determined threshold), like graphs based on real data. To do that, we randomly generate the nodes and edges of a graph $\mathcal{G}$ based on Poisson distributions, and we rely on a set of source patterns, that are randomly modified and embedded in $\mathcal{G}$ (see Fig. 1). These modified patterns are themselves temporal semantic graphs. Besides, given a global number of nodes in $\mathcal{G}$, these nodes are progressively and randomly allocated into the successive time instances

of $\mathcal{T}$, until no enough nodes are available. Our proposition relies on the approach described by [17]. The main difference is that we use different types of relations.

The generation of the random TS-graph $\mathcal{G}$ including transformed patterns, must be done in several steps in order to separate the pattern insertion process from the generation process of the rest of the graph. We therefore propose a method organised into three major steps, that are repeated for each time instance t of the graph:

- **Step 1**: Selection of a source pattern, that is transformed, and inserted in $\mathcal{G}$ starting from the current time instance (*i.e.* the first time instance of the pattern is t).
- **Step 2**: Random generation of nodes (e_j, t) associated to the current time instance, $j \in [1, m]$.
- **Step 3**: Random generation of the edges and their relation label, between pairs of nodes $((e_j, t), (e_k, t))$ for *semantic* relations, and pairs $((e_j, t), (e_l, t-1))$ for *temporal semantic* and *filiation* relations, $j, k, l \in [1, m]$.

We have chosen to put pattern insertion as step 1 because of algorithmic purposes. Indeed, by doing this way, we can adjust the number of random nodes generated in step 2 to respect as much as possible the given Poisson distribution. Conversely, when the graph construction stops, with the last time instance being t, there can be nodes (from a previously inserted pattern) in a time instance greater than t: these nodes have to be deleted.

Algorithm 1 is the main algorithm for the generation of random TS-graphs with embedded patterns. The input parameters, some regarding the patterns, some the general TS-graph, are described in Table 1.

Parameter	**Description**
λ_n	Mean of the zero-truncated Poisson distribution of the total number of nodes in the graph
λ_r	Mean of the zero-truncated Poisson distribution of the number of nodes per time instance
Λ_e	Table of three cases grouping means of the Poisson distribution of the number of each type of relations per node
$labels_n$	List of available labels for nodes
$labels_e$	Table of three lists of labels available for each type of edge
K_p	Pattern insertion probability: Probability that a source pattern is selected for insertion at a time instance
λ_t	Mean of the Poisson distribution of the number of transformations to be performed on a selected source pattern
$Pool_p$	Set of source patterns

Table 1. Parameters of Algorithm 1

To determine the total number of nodes in the graph as well as the number of nodes for each time instance, the algorithm calls for two random distributions by a zero-truncated Poisson law [8] of respective means λ_n and λ_r.

Algorithm 1: random generation of TS-graph

Data: $\lambda_n, \lambda_r, \Lambda_e, labels_n, labels_e, K_p, \lambda_t, Pool_p$
Result: random TS-graph

G=new graph
$time = 1$
// $G\_nbNodes$ is the number of nodes to generate in G
$G\_nbNodes$=random from ZT-Poisson(λ_n)
while $G\_nbNodes > 0$ **do**
  // $t\_nbNodes$ is the number of nodes to generate at current time
  $t\_nbNodes$=random from ZT-Poisson(λ_r)
  /* *STEP 1: pattern selection, transformation and insertion */
  if *toss biased coin with* K_p *probability == True* **then**
    select a source pattern from $Pool_p$
    $nbTrans$=random from Poisson(λ_t)
    randomly transform selected pattern with $nbTrans$ transformations
    $temp$=number of nodes in the transformed pattern
    if $G\_nbNodes >= temp$ **then**
      $G\_nbNodes = G\_nbNodes - temp$
      insert randomly transformed pattern from $time$ instance
    end
  end
  /* *STEP 2: random generation of nodes at $time$ instance */
  $t\_nbNodes = t\_nbNodes -$ number of patterns nodes (e, t) with $t = time$
  i=1
  if $G\_nbNodes < t\_nbNodes$ **then**
    Delete nodes (e, t) where $t \geq time$
  else
    while $(i \leq t\_nbNodes)$ **do**
      $chosenLabel$=random pick from list $labels_n$
      $newNode$=new node with $chosenLabel$ as label, $time$ as time
      add $newNode$ to graph G
      $G\_nbNodes$=$G\_nbNodes$-1
      $i = i + 1$
    end
    /* *STEP 3: random generation of relations at $time$ instance */
    foreach *node in G at time* **do**
      // number of relations starting from current node
      $nbSemanticRel$=random from Poisson(Λ_e[0])
      $nbTempSemanticRel$=random from Poisson(Λ_e[1])
      $nbFiliationRel$=random from Poisson(Λ_e[2])
      genRelation(G,$time$,$node$,$labels_e$[0],$nbSemanticRel$,$semantic$)
      if $time > 1$ **then**
        genRelation(G,$time$,$node$,$labels_e$[1],$nbTempSemanticRel$,$temporal\_semantic$)
        genRelation(G,$time$,$node$,$labels_e$[2],$nbFiliationRel$,$filiation$)
      end
    end
    $time = time + 1$
  end
end
return G

The choice of a Poisson law rather than a normal distribution is justified by the need for discrete values for the number of nodes. Moreover, the zero-truncated Poisson law is limited to 1 and, as the classical Poisson law, it is particularly accurate for small means. These characteristics are suitable for our problem, because we seek to achieve the most natural random distribution possible, and especially to avoid extreme values. Besides, it is important to specify that the number of time instances is not directly configurable, since it is decided according to the distribution of the two laws of zero-truncated Poisson mentioned above.

Furthermore, note that source patterns of $Pool_p$, can be either randomly generated beforehand by using Algorithm 1 without lines 8 to 18, as the generation of a source pattern can be treated as the generation of a subgraph, or provided by a third party.

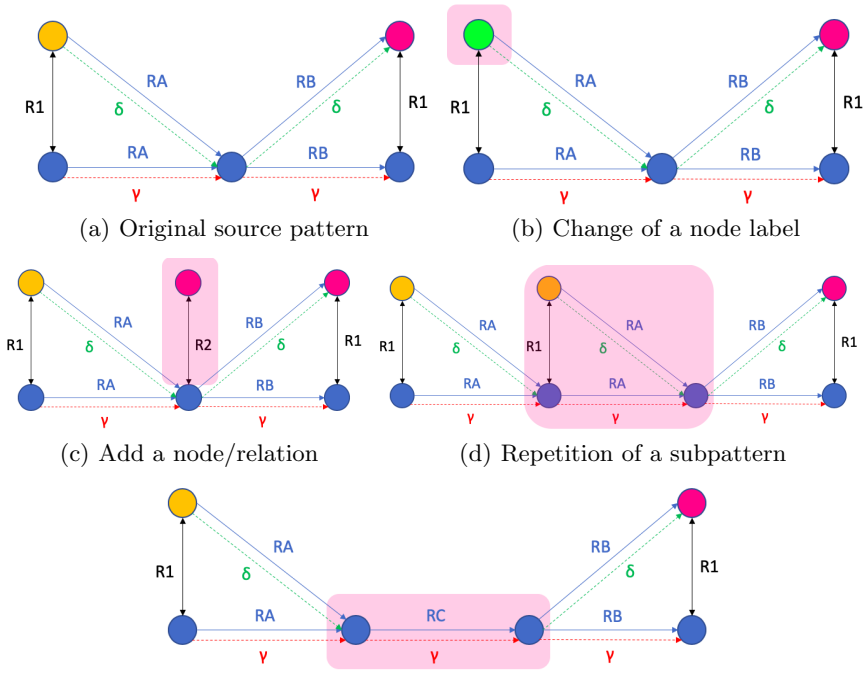
Step 1 (from lines 8 to 18 of Algorithm 1). The first step is to choose randomly, according to parameter $K_p \in [0; 1]$, uniformly, if a pattern must be inserted or not. Then a source pattern is selected in the pool of source patterns $Pool_p$. A number of transformations ($nbTrans$) are then applied to this pattern. This last number is obtained by applying a Poisson distribution with mean λ_t. This number must be small with respect to the size of the source pattern to insure that more or less the same pattern is inserted. It should be noted that if, by chance, the final graph contains similar patterns that were not intentionally incorporated, it does not matter because our final aim is to check the frequency of patterns deriving from each source pattern. The possible transformations are the following:

(i) Change of a node/relation label (*cf.* Fig. 3(b)).
(ii) Add/Remove of a node/relation (*cf.* Fig. 3(c)).
(iii) Repetition of a subpattern (*cf.* Fig. 3(d)).
(iv) Add/Remove the nodes (and their edges) associated to an intermediate time instance (*cf.* Fig. 3(e)).

The source pattern transformation algorithm is not presented in this paper. It ensures that, for each transformation, the constraints of TS-graph definition are respected (see Sect. 3). Before embedding the transformed pattern, the algorithm checks whether it is not too large, *i.e.* if its number of nodes *temp* is lesser than the number of nodes to be inserted in the graph ($G\_nbNodes$). In this case, the pattern is inserted into the graph and $G\_nbNodes$ is updated by substracting the pattern number of nodes. The pattern insertion is such that its lower time instance equals *time* (lines 13-17 of Algorithm 1).

Step 2 (from line 19 to 31 of Algorithm 1). This step consists in generating the number of missing nodes to reach the number of nodes ($t\_nbNodes$) computed for the time instance *time* by the zero-truncated Poisson law.

Inserted patterns (Step 1 of Algorithm 1) usually contain several time instances. Therefore, before adding nodes with a given time instance, it is necessary to take into account the nodes belonging to patterns inserted before (line



(a) Original source pattern (b) Change of a node label

(c) Add a node/relation (d) Repetition of a subpattern

(e) Add of an intermediate time instance

Fig. 3. Example of possible transformations of a source pattern. Changes are highlighted by a slightly colored rectangle in pink

20 of Algorithm 1). Then, the algorithm checks whether the total number of remaining nodes to be inserted ($G\_nbNodes$) is greater than the number of nodes to be inserted within the current time instance *time* ($t\_nbNodes$). If not, all existing nodes with a time instance equal or greater than *time* are to be deleted (they correspond to previously inserted patterns, that may exceed the current time instance) (line 23 of Algorithm 1). These node deletions take place only at the end of the graph construction, to prevent last time steps having a low number of nodes.

Moreover, a label is randomly assigned to each generated node.

Step 3 (from line 32 to 43 of Algorithm 1). This third and last step allows to generate and label the relations between the nodes generated for the current and the previous time instances. In the context of TS-graphs, *semantic* relations connect two nodes with the same time instance t, while *semantic temporal* and *filiation* relations connect two nodes with successive time instances $t-1$ and t.

The method must respect the uniqueness of each relation type between two nodes. So, we get, at most, a full mesh for each type of relation. Just like the generation of the nodes (see Step 2), the generation of the relations is configurable, by setting the means of three Poisson (random) distributions. Unlike the

generation of nodes, in this case, we accept that the Poisson law produces the number 0. That being said, depending on the wanted type of graph, at this step, it is possible to use other laws, for example, power law [6] in order to create TS-graphs composed of scale-free graphs.

The pseudo-code of this step is presented in Algorithm 2 and input parameters in Table 2. This algorithm generates the set of relations of a given type for a node. It adds, at most, a given number of relations. Indeed, if the node is already connected to all the other nodes of the current time instance *time* (for semantic relations) or of the time instance $time - 1$ (for temporal semantic or filiation relations), the algorithm stops the generation of the corresponding relation for this node (line 15 of Algorithm 2). Moreover, if the node belongs to an inserted pattern and has already a certain number of relations, the algorithm subtracts this number from the number of relations to generate ($nbOfRel$) (line 2 of Algorithm 2).

At the end of this step, the patterns previously added (see Step 1) are connected to the nodes generated for $time = t$ instance. This construction is correct since two nodes of the same time instance are connected with at most one semantic relation (lines 4-5 of Algorithm 2) and two nodes of successive time instances are connected with at most one temporal semantic and one filiation relations (lines 6-7 of Algorithm 2).

Parameter	**Description**
G	Graph in which, relations must be added
n	Node from which, relationships must be generated
$labels_e$	Table of three lists of labels available for each type of relationship
$nbRel$	Maximum number of relations having type $relType$ to add to node n
$relType$	Type of relations that will be generated (semantic, temporal_semantic, filiation)

Table 2. Input parameters of Algorithm 2.

In the worst case, the complexity of the main algorithm would be $O(n^2)$, n being the number of nodes. But since edges exist only between nodes within the same or two following time instances, the complexity is $O(n^2/T)$, T being the number of time instances.

Besides, there could be some inconsistencies within the graph, due to the domain semantic. For instance, let consider three nodes $(P_1; t_1)$, $(P_2; t_1)$ $(P_3; t_1)$ and their relations $(P_1; t_1)$ TPP $(P_2; t_1)$, $(P_2; t_1)$ TPP $(P_3; t_1)$, and $(P_3; t_1)$ TPP $(P_1; t_1)$: this subgraph is inconsistent wrt the RCC8 framework (TPP, *is a tangential proper part*, is a transitive and not reflexive relation). Consistency in this framework can be checked with available tools, e.g. [9], based on constraint satisfaction problem approaches.

Algorithm 2: generation of a given type of relation for a node

Data: $G, n, labels_e, nbRel, relType, time$

```
// In case node is from inserted pattern
nbRel = nbRel − number of relType relations already starting from node n;
for i=0 to nbRel do
    if relType == semantic then
        nodeList=pick all the nodes from the same time instance not already
          connected to n;
    else
        nodeList=pick all the nodes from the previous time instance not
          already connected to n;
    end
    if size of nodeList > 0 then
        chosenLabel=random pick from list labels_e
        chosenNode=random pick from list nodeList
        newRel=new relation of type relType between chosenNode and n
          with chosenLabel as label
        add newRel to graph G;
    else
        break;
    end
end
```

5 Insertion Quality Score

In this section, we introduce the *Insertion Quality Score (IQS)* that evaluates how patterns are embedded within the generated graphs.

Our aim is to generate random TS-graphs controlled by a certain number of parameters, which can be extracted from graphs based on real data. Indeed, the objective is to provide data sets for data mining methods applying to these types of graphs, and particularly for the search of frequent patterns. In order to test these methods efficiently, it is necessary to produce graphs at different levels of complexity. The complexity can be assessed by a combination of different parameters affecting the inserted patterns, regarding the number of nodes and the number of the different types of edges, with respect to the same characteristics in the general graph.

We first introduce some measures used to determine the Insertion Quality Score. To do this, we first define some notations. Let $\mathcal{G}_1$ be a TS-graph, with V the set of nodes and $E = E_\Sigma \cup E_\Phi$ the set of edges. Let $V_P \subset V$ be the set of all nodes belonging to an inserted pattern. $\theta = |\mathcal{T}|$ is the total number of times intances in $\mathcal{G}_1$ and θ_P is the number of all time instances associated with nodes of V_P. The corresponding set is defined as follows $\mathcal{T}_P = \{t \in \mathcal{T} | \exists (e_i, t) \in V_P\}$. Following this idea, we define $V_{\mathcal{T}_P} = \{(e_i, t_j) \in V | t_j \in \mathcal{T}_P\}$ i.e. the set of all nodes associated to time instances of $\mathcal{T}_P$.

Then $\epsilon = |E|$ is the total number of edges, $\nu = |V|$ the total number of nodes in $\mathcal{G}_1$; $\nu_P = |V_P|$ is the number of all the nodes belonging to an inserted pattern;

$\nu_{\mathcal{T}_P} = |V_{\mathcal{T}_P}|$ is the number of nodes associated to time instances of $\mathcal{T}_P$; ϵ_P is the number of edges linking nodes of V_P. We denote $\epsilon_{\bar{P}} = \epsilon - \epsilon_P$, $\nu_{\bar{P}} = \nu - \nu_P$, $\theta_{\bar{P}} = \theta - \theta_P$, $\nu_{\mathcal{T}_{\bar{P}}} = \nu - \nu_{\mathcal{T}_P}$. Furthermore, the subset of temporal semantic (*resp.* semantic, filiation) edges is noted E^{ts} (*resp.* E^s,$E^f = E_\Phi$), and its cardinality ϵ^{ts} (resp. ϵ^s, ϵ^f).

Four measures can then be defined, for taking into account the 3 types of edges and for the nodes.

- $Me_s \in [0;1]$ (Formula 1 with $i = s$) allows to measure the proportionality value between the average number of semantic edges belonging to an inserted pattern and the average number of semantic edges not belonging to an inserted pattern.
- $Me_{ts} \in [0;1]$ (Formula 1 with $i = ts$) is like Me_s but for temporal semantic edges.
- $Me_f \in [0;1]$ (Formula 1 with $i = f$) is like Me_s but for filiation edges.

$$Me_i = \frac{\min(\frac{\epsilon_P^i}{\nu_P}, \frac{\epsilon_{\bar{P}}^i}{\nu_{\bar{P}}})}{\max(\frac{\epsilon_P^i}{\nu_P}, \frac{\epsilon_{\bar{P}}^i}{\nu_{\bar{P}}})} \qquad \text{with } i \in \{s, ts, f\} \tag{1}$$

  In practice, the more Me_i with $i \in \{s, ts, f\}$ is close to 1, the more the average number of a type of edges in patterns is close to the average number of the same type of edges in the rest of the graph. This means that patterns are well embedded, regarding this type of edges.
- $M_n \in [0;1]$ (Formula 2) represents the proportionality value between the average number of nodes for the time instances of $\mathcal{T}_P$ (t1 and t2 in Fig. 4) and the average number of nodes for the time instances of $\mathcal{T}_{\bar{P}}$ (t3 and t4 in Fig. 4).

$$M_n = \frac{\min(\frac{\nu_{\mathcal{T}_P}}{\theta_P}, \frac{\nu_{\mathcal{T}_{\bar{P}}}}{\theta_{\bar{P}}})}{\max(\frac{\nu_{\mathcal{T}_P}}{\theta_P}, \frac{\nu_{\mathcal{T}_{\bar{P}}}}{\theta_{\bar{P}}})} \tag{2}$$

  In practice, the more M_n is close to 1, the less nodes coming from inserted patterns do modify the average number of nodes per time instance in the graph. In other words, if M_n is close to 1, this amounts that patterns are well embedded in the graph, regarding the temporal distribution of nodes. For example, in Fig. 4, $\frac{\nu_{\mathcal{T}_P}}{\theta_P} = \frac{4+4}{2} = 4$ and $\frac{\nu_{\mathcal{T}_{\bar{P}}}}{\theta_{\bar{P}}} = \frac{3+5}{2} = 4$, and thus $M_n = 1$.

Finally the Insertion Quality Score of a graph is defined as a linear combination of the previous measures, to give a global glance of the pattern insertion quality in this graph $\mathcal{G}$.

$$\begin{aligned} &IQS(\mathcal{G}) = aMe_s + bMe_{ts} + cMe_f + dM_n \\ &\text{with } a + b + c + d = 1 \end{aligned} \tag{3}$$

The interpretation of Formula 3 is as follows:

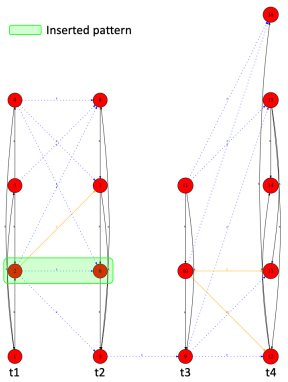


Fig. 4. Example of an inserted pattern for M_n metric

- If $IQS(\mathcal{G}) \to 0$, then, on average, the inserted patterns are much different from the average of the rest of the graph, from a structural and semantical point of view (nodes and type of edges). We can say that the patterns are badly inserted and thus easily identifiable.
- If $IQS(\mathcal{G}) \to 1$, then the inserted patterns are, on average, close to the average of the rest of the graph. The patterns are perfectly inserted, so the difficulty in locating patterns is consequent.

It is important to note that the IQS indicates an *average score*. For example, trying to insert patterns from a great variety of source patterns will result in a loss of accuracy of the IQS. In this case it would be interesting to compute an IQS for the subset of inserted patterns associated to each source pattern in order to have an idea of their insertion quality. Indeed, our final goal being to test methods of frequent pattern mining, what interests us in priority is to measure the difficulty in finding patterns. However, other metrics can be used, especially when the aim is to generate graphs close to real graphs (e.g. temporal node centrality [16]).

6 Conclusion and future works

We have defined a model for *temporal semantic* graphs (TS-graph), which allows to represent the temporal evolution of entities linked by semantical relations. Moreover, the lack of this type of data in the literature has led us to propose an algorithm for the generation of random TS-graphs. Given temporal semantic patterns are randomly modified and inserted into a graph which nodes and edges are randomly generated.

The generation of TS-graphs is fully configurable, allowing them to be close to data-based graphs by using different laws adapted to the reality. Moreover, we have proposed a measure, namely the *Insertion Quality Score* (IQS), to evaluate the quality of patterns insertion. In order to refine this evaluation, the IQS could be extended to take into account more information on nodes or relation labels.

The presented algorithms have been implemented in Python 3.6. Our aim is to generate graphs with large time sequences and / or with large spatial extensions. Experiments are currently conducted to study the effects of input parameters on the performances of our algorithms and the characteristics of resulting graphs.

In the future, we will focus on particular applications (e.g. modeling the activity of brain regions or the evolution of land-use in agricultural landscapes). A major issue will be to verify the graph consistency, which is application-dependent. Furthermore, it would be interesting to extend this work by preserving local structures found in real graphs, as proposed in [22], in order to make the generated graphs closer to these real graphs. Our approach for the generation of random graphs will finally be used to evaluate methods of frequent spatio-temporal patterns mining by comparing the patterns found by the different methods with respect to those inserted in the graph.

References

1. Aiello, W., Chung, F., Lu, L.: A random graph model for massive graphs. In: Proc. of the 32th ACM symposium on Theory of computing, STOC'00. pp. 171–180 (2000)
2. Batagelj, V., Brandes, U.: Efficient generation of large random networks. Physical Review E **71**(3), 036113 (2005)
3. Bollobás, B.: Random graphs. No. 73 in Cambridge Studies in advanced mathematics (2001)
4. Bonifati, A., Holubová, I., Prat-Pérez, A., Sakr, S.: Graph generators: State of the art and open challenges. ACM Comput. Surv. **53**(2) (Apr 2020). https://doi.org/10.1145/3379445
5. Campbell, C., Shea, K., Albert, R.: Comment on "control profiles of complex networks". Science **346**(6209), 561–561 (2014)
6. Ferrer i Cancho, R., Solé, R.V.: Optimization in complex networks. In: Statistical mechanics of complex networks, pp. 114–126 (2003)
7. Chakrabarti, D., Zhan, Y., Faloutsos, C.: R-mat: A recursive model for graph mining. In: Proceedings of the 2004 SIAM Int Conference on Data Mining. pp. 442–446 (2004)
8. Cohen, A.C.: Estimating the parameter in a conditional poisson distribution. Biometrics **16**(2), 203–211 (1960)
9. Condotta, J.F., Ligozat, G., Saade, M.: A Generic Toolkit for n-ary Qualitative Temporal and Spatial Calculi. In: The 13th Int Symposium on Temporal Representation and Reasoning (TIME'06). pp. 78–86. Budapest, Hungary (2006)
10. Del Mondo, G., Rodríguez, M., Claramunt, C., Bravo, L., Thibaud, R.: Modeling consistency of spatio-temporal graphs. Data & Knowledge Engineering **84**, 59 – 80 (2013)
11. Del Mondo, G., Stell, J.G., Claramunt, C., Thibaud, R.: A graph model for spatio-temporal evolution. Journal of Universal Computer Science **16**, 1452–1477 (2010)
12. Erdös, P., Rényi, A.: On random graphs. Publ. Math. Debrecen **6**, 290–297 (1959)
13. Hadian, A., Nobari, S., Minaei-Bidgoli, B., Qu, Q.: Roll: Fast in-memory generation of gigantic scale-free networks. In: Proceedings of the 2016 Int Conference on Management of Data. pp. 1829–1842 (2016)

14. Jahel, C., Baron, C., Vall, E., Karambiri, M., Castets, M., Coulibaly, K., Bégué, A., Seen, D.L.: Spatial modelling of agro-ecosystem dynamics across scales: A case in the cotton region of West-Burkina Faso. Agricultural Systems **157**, 303 – 315 (2017)
15. Kaiser, M., Martin, R., Andras, P., Young, M.P.: Simulation of robustness against lesions of cortical networks. European Journal of Neuroscience **25**(10), 3185–3192 (2007)
16. Kim, H., Anderson, R.: Temporal node centrality in complex networks. Phys. Rev. E **85**, 026107 (Feb 2012). https://doi.org/10.1103/PhysRevE.85.026107
17. Kuramochi, M., Karypis, G.: Frequent subgraph discovery. In: Proceedings 2001 IEEE Int Conference on Data Mining. pp. 313–320 (2001)
18. Leskovec, J., Chakrabarti, D., Kleinberg, J., Faloutsos, C., Ghahramani, Z.: Kronecker graphs: An approach to modeling networks. Journal of Machine Learning Research **11**(Feb), 985–1042 (2010)
19. Leskovec, J., Chakrabarti, D., Kleinberg, J., Faloutsos, C.: Realistic, mathematically tractable graph generation and evolution, using kronecker multiplication. In: European conference on principles of data mining and knowledge discovery. pp. 133–145. Springer (2005)
20. Nica, C., Braud, A., Le Ber, F.: Exploring Heterogeneous Sequential Data on River Networks with Relational Concept Analysis. In: 23rd International Conference on Conceptual Structures, Edimbourg, United Kingdom. LNAI, vol. 10872, pp. 152–166. Springer (2018)
21. Park, H., Kim, M.S.: Trilliong: A trillion-scale synthetic graph generator using a recursive vector model. In: Proceedings of the 2017 ACM Int Conference on Management of Data. pp. 913–928 (2017)
22. Purohit, S., Holder, L.B., Chin, G.: Temporal graph generation based on a distribution of temporal motifs. In: 14th International Workshop on Mining and Learning with Graphs (MLG'18) (2018)
23. Randell, D.A., Cui, Z., Cohn, A.G.: A Spatial Logic based on Regions and Connection. In: Proceedings 3rd Int Conference on Knowedge Representation and Reasoning (1992)
24. Rossetti, G.: RDYN: graph benchmark handling community dynamics. Journal of Complex Networks **5**(6), 893–912 (2017)
25. Wittemyer, G., Keating, L.M., Vollrath, F., Douglas-Hamilton, I.: Graph theory illustrates spatial and temporal features that structure elephant rest locations and reflect risk perception. Ecography **40**(5), 598–605 (2017)
26. Yuan, J., Li, X., Zhang, J., Luo, L., Dong, Q., Lv, J., Zhao, Y., Jiang, X., Zhang, S., Zhang, W., Liu, T.: Spatio-temporal modeling of connectome-scale brain network interactions via time-evolving graphs. Neuroimage **180**, 350–369 (2018)