On the Synthetic Generation of Semantic Web Schemas

Yannis Theoharis[1,2], George Georgakopoulos[2], and Vassilis Christophides[1,2]

Institute of Computer Science, FORTH, Vassilika Vouton, P.O.Box 1385, GR 71110, Heraklion, Greece
Department of Computer Science, University of Crete, P.O.Box 2208, GR 71409, Heraklion, Greece
{theohari,christop}@ics.forth.gr,
ggeo@csd.uoc.gr

Abstract. In order to cope with the expected size of the Semantic Web (SW) in the coming years, we need to benchmark existing SW tools (e.g., query language interpreters) in a credible manner. In this paper we present the first RDFS schema generator, termed PoweRGen, which takes into account the morphological features that schemas frequently exhibit in reality. In particular, we are interested in generating synthetically the two core components of an RDFS schema, namely the *property* (relationships between classes or attributes) and the *subsumption* (subsumption relationships among classes) graph. The total-degree distribution of the former, as well as the out-degree distribution of the Transitive Closure (TC) of the latter, usually follow a `power-law`. PoweRGen produces synthetic property and subsumption graphs whose distributions respect the `power-law` exponents given as input with a confidence ranging between $90 - 98\%$.

1 Introduction

Semantic Web (SW) [4] gains increasing popularity nowadays. A significant amount of RDF/S schema and instance descriptions are already available on the WWW. According to [3], 16.69% of XML documents rely on RDF/S [6], while their volume reaches the 31.71% of the available XML data. As the size of the SW is expected to be further increased in the coming years, the scalability of SW storage, query or update tools becomes crucial [24]. Following the tradition of data management community, new benchmarks and synthetic data generators need to be developed for the SW. This need is also underlined by the recent RDF/S benchmarking initiatives undertaken by the W3C consortium [28].

Unlike XML data generators [21,2], which rely on fixed DTDs or XML schemas, we pay particular attention to the synthetic generation of SW schemas specified in RDFS [6]. This is motivated by the fact that advanced SW reasoning functionalities (e.g., concerning subsumption relationships) can be built only by exploiting the additional information encoded in these schemas.

Our focus in RDFS, instead of OWL, schemas is motivated by the fact that the majority of available SW schemas rely on the RDFS specification (according

V. Christophides et al. (Eds.): SWDB-ODBIS 2007, LNCS 5005, pp. 98–116, 2008.
© Springer-Verlag Berlin Heidelberg 2008

to [9], 85.45% relies on RDFS, while 14.55% on OWL). Additionally, the core RDFS features are also exploited by OWL. Therefore, our work can be seen as a basis that can be extended to OWL schema generation.

RDFS schemas are essentially graphs whose arcs are of different nature, namely, a) arcs representing subsumption relationships among classes, and b) arcs representing relations between classes (e.g., *has_a*) or attributes (e.g., *title*), collectively called properties. In this context, for each RDFS schema we essentially need to generate two graphs that have the same set of nodes (i.e., classes or literal types), namely, the *subsumption*, and the *property* graph.

It is well known that a `power-law` is a function of the form $f(x) \propto x^{-b}$. The *total-degree* distribution of the property graph, as well as the *out-degree* (i.e., the class descendants) distribution of the Transitive Closure (TC) of the subsumption graph usually follow a `power-law` distribution [25]. Furthermore, classes that appear as domain of many properties are located highly in the class hierarchies, i.e., classes with high out-degree in the *property* graph are located at the first levels of the *subsumption* graph. Thus, synthetic RDFS schemas should exhibit similar features with those frequently encountered in reality.

In this paper we propose the first synthetic RDFS schema generator, termed PoweRGen, which takes as input: a) the number of schema classes and properties, b) the characteristic exponents of the aforementioned `power-law`s, c) the depth of the subsumption graph, and d) the knowledge whether the subsumption graph should be a DAG or a tree. The property graph of the generated schemas follows the `power-law` given as input (98% confidence). The same is also true for the subsumption graph in the case of trees. For DAGs, the generated subsumption graph approximates with a 90.3% confidence a `power-law`, whose characteristic exponent follows (99.4% confidence) the one given as input.

The remainder of this paper is organized as follows: Section 2 introduces the main features of the *property* and *subsumption* graph forming an RDFS schema. Section 3 presents the generation of RDFS schemas. Section 4 presents the results of an experimental evaluation, while Section 5 compares our graph generation method with related work. Finally, Section 6 identifies issues for future research.

2 Semantic Web Schema Graphs

RDFS schemas are usually represented as directed labeled graphs, whose nodes are classes or literal types and arcs are properties. These graphs may have self-loops (representing recursive properties) and multiple arcs (when two classes are connected by several properties). The leftmost part of Figure 1 depicts an example of a schema. In particular, SW schemas have two different kinds of arcs: subsumption arcs (*rdfs:subclassOf*), and user defined ones. The former comprises *subsumption* relationships among classes (which are transitive in nature), while the latter comprises attributes or relationships among classes, which are called *properties*. As the interpretation of these two arc kinds is different, for each RDFS schema we need to define two graphs: a) the *property* (e.g., the second part of Figure 1) and b) the *subsumption* graph (e.g., the third part of Figure 1).

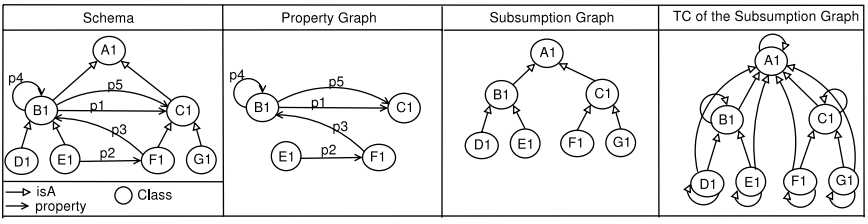


Fig. 1. A SW schema and its constituent graphs

Both graphs have the same set of nodes (i.e., the union of classes and literal types used in the schema) but they comprise different kinds of arcs.

Definition 1. *The* property graph *of a schema is a directed graph* $\mathcal{G}_p = (\{C \cup L\}, P)$*, where* C *is a set of nodes labeled with a class name,* L *is a set of nodes labeled with a literal type,* P *is a set of arcs of the form* $\langle c_1, p, c_2 \rangle$ *where* $c_1 \in C$*,* $c_2 \in C \cup L$*, and* p *is a property name.*

Definition 2. *The* subsumption graph *of a schema is a directed graph* $\mathcal{G}_s = (C, P_s)$*, where* C *is a set of nodes labeled with a class name and* P_s *is a binary relation over the elements of* C*.*

By considering the RDFS semantics [6] we can obtain the TC of the subsumption graph of a schema (e.g. the rightmost part of Figure 1).

Definition 3. *Let* $\mathcal{G}_s = (C, P_s)$ *be the subsumption graph of a schema. The* TC of the subsumption graph*, denoted by* $\mathcal{G}_s^*$*, is the pair* (C, P_s^*) *where* P_s^* *is the* TC *of* P_s*.*

It should be stressed that, according to the RDF/S semantics [14], class subsumption is a transitive relation and hence we should study the TC of the subsumption graph. The effect of the subsumption to the property graph is implicit.

Concerning the *property* graph, the out-(resp. in-) degree of a class is the frequency of occurrence of this class in the ranges (resp. domains) of properties, while the total-degree of a class is the frequency of occurrence of this class in either the ranges or the domains of properties. Furthermore, the out-(resp. in-) degree of a class in the TC of the *subsumption* graph corresponds to transitive subclasses (resp. superclasses) of this class.

2.1 RDFS Schema Graph Features

In this Section, we briefly recall the experimental results of [25] highlighting the main graph features exhibited by an adequate large corpus[1] of big sized schemas available on the SW. In particular, we are interested on the core RDFS features

[1] For more information see http://athena.ics.forth.gr:9090/RDF/VRP/SWSchemas/

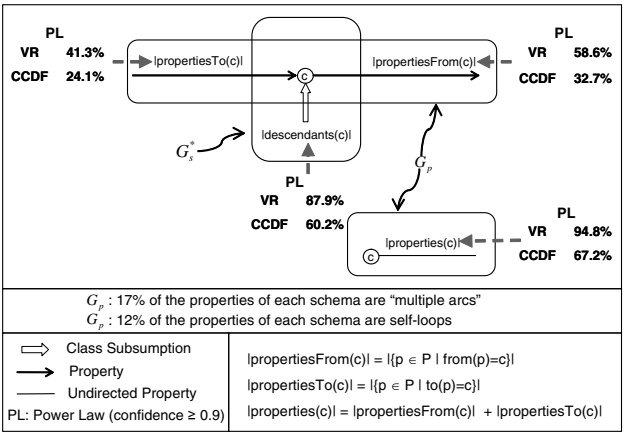


Fig. 2. Main graph features of SW schemas

which are also exploited by OWL. We report the degree distributions of the property and subsumption graph. Additionally, we report combinatorial findings concerning the two graphs (e.g., the class distribution per level of the subsumption hierarchy, as well as, the number of related properties) which allows us to sketch a more accurate picture about the morphology of real RDFS schemas.

Degree Distributions. In order to analyze the degree distributions of schema graphs, 4 (mainly) Discrete Random Variables (DRV) were considered. In particular, 3 of them corresponded to the in-/out-/total-degrees of the *property* graph, $\mathcal{G}_p$, and the fourth to the out-degrees (corresponding to class descendants) of the TC of the *subsumption* graph, $\mathcal{G}_s^*$. Possible `power-law`s were investigated on two functions related to each DRV:

i) complementary cumulative probability density function (CCDF), i.e., $P(X \geq x)$
ii) the relationship among the values of its range set, denoted by D, and their rank in decreasing order (VR), i.e., a function $f : [1, ..., |D|] \rightarrow D$.

Implicitly, probability density function (PDF), i.e., $P(X = x)$, was also considered, since CCDF and PDF are essentially the same and whenever PDF follows a `power-law` with exponent b, CCDF follows a `power-law` with exponent $b - 1$ [7].

The results of the analysis of [25] are summarized in Figure 2. The main conclusion drawn was that the majority of RDFS schemas approximate a `power-law` for the total-degree functions (94.8% for VR and 67.2% for CCDF) of the property graph, while for out- and in-degrees the corresponding percentages are significantly lower. The characteristic `power-law` exponents for the total-degree *VR* (resp. CCDF) lie in $[0.79, 2.18]$ (resp. $[0.65, 2.05]$). Concerning the subsumption graph, the out-degree VR (resp. CCDF) approximate a `power-law` for the 87.9% (resp. 60.2%) of RDFS schemas. The characteristic exponents in this case lie in

$[0.97, 2.44]$ (resp. $[0.54, 1.47]$) for VR (resp. CCDF). Class ancestors distribution was not observed to follow a `power-law` (nor any other) distribution for real SW schemas.

Morphological Features. An interesting observation that correlates the property graph with the subsumption graph is that most properties have as domain, classes which are located highly in the class hierarchy, i.e., somewhere between the root and the middle level. This fact shows that the specification of a class in RDFS schemas is used more for classification purposes rather than for refining classes with additional properties. The same trend was observed for range classes, although the dominance of classes located higher over those located lower in the subsumption graph is not so important as for property domains.

Furthermore, the consideration of one additional DRV corresponding to number of classes per hierarchy level revealed the skewness of the distribution of classes per level. Specifically, the VR function approximates (for the 42.1% of real RDFS schemas) a `power-law` while the PDF always approximates the uniform distribution. Moreover, the corresponding exponent of the VR function approximately depends linearly on the number of nodes (classes). In addition, the level, denoted by k, at which the maximum number of nodes are located is approximately $0.75 \times depth$ for real RDFS schemas, while the number of classes located at a specific level decreases as long as its distance to k increases. Last, on average the 75% of classes of real RDFS schemas are leaves.

Finally, there exists a strong correlation between the out- and in-degrees of nodes of $\mathcal{G}_p$. In other terms, classes that appear as domain of many properties appear also as range of many (the same or other) properties. Last but not least, the percentage of self-loops (i.e., recursive properties) is quite significant, i.e., 12.6% in the average case. The average percentage of multiple arcs, that is properties that share the same domain and range, is slightly higher, i.e., 17.7%.

3 Synthetic SW Schema Generation

In this Section, we present the main algorithmic steps of the `power-law` based RDFS synthetic schema generator (PoweRGen). In particular, PoweRGen comprises the:

i) generation of the total-degree (resp. out-degree) sequence of $\mathcal{G}_p$ (resp. $\mathcal{G}_s^*$). To this end, it exploits two methods to sample a bag of values from a DRV that follows a `power-law`,
ii) generation of the property (resp. subsumption) graph by using $GenerateG_p$ (resp. $GenerateG_s$) algorithms. These algorithms are based on *Linear Programming* and exploit features that schemas frequently exhibit in reality (see Section 2.1)
iii) mapping of nodes of $\mathcal{G}_p$ to nodes of $\mathcal{G}_s^*$ in order to construct the output schema.

In the sequel, we elaborate on each one of the above three steps.

3.1 Sampling a DRV That Follows a PL

To generate the total-degree sequence of $\mathcal{G}_p$, as well as the out-degree sequence of $\mathcal{G}_s^*$. we exploit two methods (see [23]) to sample a bag of values from a DRV that follows a `power-law`. The former implements well known ideas and is independent of the nature of the PDF function. The latter applies only when the VR function is a `power-law`. The former is useful in the case that the DRV is bounded by a maximum value. This is actually the case of out-degrees of $\mathcal{G}_s^*$ in which the maximum allowed value is $|C|-1$ (i.e., root descendants). The latter is useful in the case that the sum of the bag of sampled values is predefined. This is actually the case of the total-degrees of $\mathcal{G}_p$, where the sum of all total-degree values should be equal to twice the number of schema properties (graph arcs).

3.2 Generating the Property Graph

Bellow, we explain in detail algorithm *GenerateG*$_p$.

Algorithm. *GenerateG*$_p$	
Inputs.	
	b: the VR exponent of the total-degree distribution,
	N_c/N_p: the number of schema classes/properties,
	p_0: percentage of classes that neither appear as domain nor as range of any property.
(1)	$D := VRSampling(b, \lfloor(1-p_0)\times N_c\rfloor, 2\times N_p)$;
	$D_{in} := \emptyset$; $D_{out} := \emptyset$;
(2)	for $i = 1$ to N_c do
	$D_{in}[i] = D_{out}[i] = \frac{1}{2} \times D[i]$;
(3)	Choose randomly nodes and attach on them a set S of $\lfloor 0.126 \times N_p \rfloor$ self-loops (modify D_{in} and D_{out});
(4)	Choose randomly pairs of nodes and attach on them a set M of $\lfloor 0.177 \times N_p \rfloor$ multiple arcs (modify D_{in} and D_{out});
(5)	$E :=$ the set of arcs of the solution of the *LP1* instance corresponding to D_{in} and D_{out};
(6)	$E_p = E \uplus S \uplus M$;

Using *VRSampling* we can generate the *total-degree* sequence, denoted by D, of the $\mathcal{G}_p$ based on the VR exponent of the total-degree distribution, denoted by b (step 1). We choose to generate D instead of *Dout* and *Din*, because the percentage of real RDFS schemas that approximate a `power-law` for the *total-degree* distribution is bigger than the corresponding percentages for *out-* and *in-degree* distributions (see Section 2.1). We should also mention that there exist classes in real SW schemas that neither appear as domain nor as range of any property, i.e., their total-degree is 0. The percentage of such classes, denoted by p_0, can be a parameter of the generator (a typical value is 50%). Hence, we generate D by setting $(b, N, sum) = (b, \lfloor(1-p_0)\times N_c\rfloor, 2\times N_p)$ in *VRSampling*.

As a next step (2), we need a method that splits D into $Dout$ and Din. This method should adhere to the fact that there exists a strong correlation between the out- and in-degrees of nodes. Additionally the sum of $Dout$ elements should be equal to the sum of Din, since it equals the number of graph arcs. A simple formula that adheres to both conditions is $\sum_{v \in V} D(v) = \sum_{v \in V} Dout(v) + \sum_{v \in V} Din(v)$.

Furthermore, we will randomly choose nodes (whose both out- and in-degree are bigger than 1) to assign them ($0.126 \times N_p$ in total) self-loops and pairs of nodes (whose both out- and in-degree are bigger than 2) to assign them ($0.177 \times N_p$ in total) multiple arcs (steps 3 and 4). Let S be the set of the chosen self-loops and M the set of the chosen multiple arcs. For each self-loop $\langle u, u \rangle$, we consider that $Dout(u) := Dout(u) - 1$ and $Din(u) := Din(u) - 1$. Similarly for each multiple arc $\langle u, v \rangle$, we consider that $Dout(u) := Dout(u) - 1$ and $Din(v) := Din(v) - 1$.

We can reduce the problem of generating a directed graph without self-loops and multiple-arcs given $Dout$ and Din as follows:

(LP1). Let G be a graph and let E be the set of its candidate arcs. The generation of G given its out- and in-degree sequences can be reduced to an LP instance of the form:

$$\begin{aligned} &min\ \mathbf{0}^{\boldsymbol{T}}\boldsymbol{x} \\ &\textstyle\sum_{(v,u) \in E} x_{v,u} = Dout(v),\ \forall v \in V \\ &\textstyle\sum_{(u,v) \in E} x_{u,v} = Din(v),\ \ \forall v \in V \\ &0 \leq x_{u,v} \leq 1, \qquad \forall \langle u, v \rangle \in E \end{aligned}$$

We consider as candidate edges all the $N \times (N-1)$ possible edges of a *directed graph* without *self-loops* and *multiple edges*. We should stress that, although we allowed the edges variables $x_{u,v}$ to take non-integer values, every solution of an *LP1* instance is integral (all the variables $x_{u,v} \in \{0, 1\}$). This is ensured by the fact that (see Theorem 1) the matrix A of every *LP1* instance is (for a proof see [23]) *totally unimodular (TUM)*, i.e., every square submatrix of A has determinant equal to 0 or ± 1. It is well known [15,27] that if A is a TUM matrix, then all the vertices of the polytope $\{\boldsymbol{x} : A\boldsymbol{x} = \boldsymbol{b},\ \boldsymbol{x} \geq 0\}$ are integer for any integer vector $\boldsymbol{b}$. *Simplex* algorithm seeks for the optimum in the vertices of the polytope defined by an LP instance. Since, every vertex of the polytope defined by an *LP1* instance is integral every solution of it is integral (in our case $x_{u,v} \in \{0, 1\}$).

Theorem 1. *Every LP1 instance can be described as* $\{\boldsymbol{x} : A\boldsymbol{x} = \boldsymbol{b},\ 0 \leq \boldsymbol{x} \leq 1\}$ *where* A *is a TUM* $m \times n$ *matrix.*

Attributes. In addition, we need to consider arcs which have as destination *Literals* (e.g., String, Integer), i.e., attributes of classes. After generating $\mathcal{G}_p$ we can add to the set of its nodes V, the Literal types, as specified in *XML schema*[2].

[2] http://www.w3.org/XML/Schema

Then we connect them to the pre-existent nodes of $\mathcal{G}_p$ under the condition that the total-degree sequence of $\mathcal{G}_p$ remains the same. This constraint can be satisfied by replacing a number k of arcs of the form $\langle u, v\rangle$, where u, v correspond to classes, such that $Dout(v) = 0$ (nodes representing literal types should have out-degree zero), with arcs of the form $\langle u, w\rangle$, where w is a node that corresponds to a *Literal* type. The number k of the attributes can be given as input (e.g., as a percentage of N_p).

Labeling properties. We should notice, that the output of $GenerateG_p$ is a bag of unlabeled edges, i.e. pairs of nodes. In the current implementation of PoweRGen, we consider a unique label (URI) per property in order to generate an RDFS schema. However, we provide the appropriate abstractions that are needed by a programmer to implement his one labeling policy. For instance, one might need the same property to have multiple domains / ranges. In that case, one should label with the same URI more than one edges.

3.3 Generating the Subsumption Graph

In order to generate $\mathcal{G}_s$ we consider as additional input the characteristic exponent, denoted by b, of the `power-law` of the PDF function of the *class descendants* distribution. Moreover, we take into account the depth, denoted by d, of $\mathcal{G}_s$ as well as the information whether $\mathcal{G}_s$ should be a DAG or a tree. Bellow, we detail $GenerateG_s$ algorithm.

Algorithm. $GenerateG_s$

Inputs.
b: the PDF exponent of the class descendants distribution,
N_c: the number of schema classes.

(1) $Dout := PDFSampling(b, N_c - 1, 0.25 \times N_c)$;
(2) for $i = 1$ to $\lfloor 0.75 \times N_c \rfloor - 1$ do
  $D_{out} := D_{out} \uplus \{0\}$;
(3) $\gamma := 0.0017 \times N_c + 1.36$;
(4) $S := VRSampling(\gamma, d, N_c)$ (Order S in descending order);
(5) $k = \lfloor \frac{3}{4} \times d \rfloor$;
(6) $L :=$ the set of $\mathcal{G}_s^*$ levels ordered according to $f(l) = |k - l|$;
(7) for $i = 1$ to d do
  for $j = 1$ to $S[i]$ do
    $Din.append(L[i])$;
(8) Order $Dout$ in descending and Din in ascending order;
(9) $E :=$ the set of arcs of the solution of the *LP2*
(or *LP3* if $\mathcal{G}_s$ is considered to be a DAG) instance
corresponding to D_{in} and D_{out};
(10) if $\mathcal{G}_s$ is a Tree, then $E_s := E$;
else $E_s :=$ the transitive reduction of E;

Generating Dout of $\mathcal{G}_s^*$. Using the *PDFSampling* method, we can generate the *out-degree* sequence of $\mathcal{G}_s^*$. Specifically, the biggest allowed value is $N_c - 1$, since the *root* node has $N_c - 1$ descendants. Furthermore, on average the 75% of classes of real RDFS schemas are leaves, i.e. their out-degree is 0. Thus, we choose the following parameters $(b, M, N) = (b, N_c - 1, 0.25 \times N_c)$ for the sampling (step 1). To obtain a sequence of length N_c we add $0.75 \times N_c - 1$ times (corresponding to the 0.75% of leaf classes) the value 0 (step 2).

Generating Din of $\mathcal{G}_s^*$. The generation of *Din* of $\mathcal{G}_s^*$ is not as easy as that of *Dout*. This is due to the fact that no frequent pattern has been observed for the distribution of *class ancestors*. The best choice in this case, is to exploit the fact that the *classes per level* VR approximates a `power-law` for a significant proportion of real SW schemas (see Section 2.1). Moreover, the characteristic VR exponent, denoted by γ, approximately depends linearly on the number of nodes (classes). In this respect, we can produce a sequence of values, which correspond to the number of nodes that are located at a specific level. To this end, we sample according to the *classes per level* VR function with parameters $(b, N, sum) = (\gamma, d, N_c)$. This is due to the fact that we want to distribute N_c nodes to d levels (steps 3 and 4).

However, we still do not know to which level a specific value of the sampled set, denoted by S, corresponds. We exploit the fact that the level, denoted by k, at which the maximum number of nodes are located is approximately $0.75 \times depth$ for real SW schemas. Let x_i be the $i-th$ biggest value of S. We order levels of $\mathcal{G}_s^*$ according to their distance to the most populated level, i.e., $L = \langle k, k+1, k-1, k+2, k-2, ... \rangle$ (step 6). Then, we consider that x_i nodes are located at level $L(i)$.

Combining Dout and Din of $\mathcal{G}_p$. If $\mathcal{G}_s$ is a tree, the level at which a node is located coincides with the number of its transitive ancestors, i.e., its in-degree in $\mathcal{G}_s^*$ (step 7). However, if $\mathcal{G}_s$ is a DAG, the level of a node consists only the maximum lower bound for its in-degree. In the sequent, we use the symbol *Din* only for trees, while symbol Din^- for DAGs.

Trees We can reduce the problem of generating a tree given the degree sequences (*Dout* and *Din*) of its TC as follows:

(LP2). Let $V_i = \{v \in V \mid Din(v) = i\}$ and d be the maximum value of i. V_i corresponds to the set of nodes located to the $i-th$ level of the tree G and d to its depth. Then the problem of generating G given the sequences of its transitive closure is reduced to the LP instance of the form:

$$
\begin{aligned}
&min\ \mathbf{0^T x}\\
&\textstyle\sum_{u \in V_{i+1}} (Dout(u) + 1) \times x_{v,u} = Dout(v),\ \ \forall i \in [0, d-1],\ \forall v \in V_i\\
&\textstyle\sum_{u \in V_{i-1}} x_{u,v} = 1,\ \ \forall i \in [1, d],\ \forall v \in V_i\\
&0 \leq x_{u,v} \leq 1,\ \ \forall \langle u, v \rangle \in E
\end{aligned}
$$

To determine the candidate arcs (*LP2* variables) of G, we observe that, an arc $\langle u, v \rangle \in V \times V$ can be an arc of a Tree $G : (V, E)$ only if, $Dout(u) > Dout(v)$

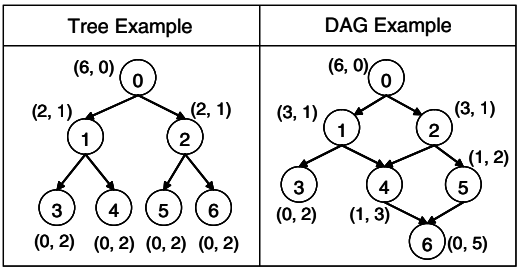


Fig. 3. An Example of a Tree (left) / DAG (right)

and $Din(u) = Din(v) - 1$. The first condition, (i.e., $Dout(u) > Dout(v)$) denotes that u should have more descendants than v and holds for every DAG (see for instance both parts of Figure 3), while the second one (i.e., $Din(u) = Din(v) - 1$) denotes that u should have one less ancestor than v and holds only for Trees. For instance, in the DAG of the right part of Figure 3, there exists an edge $\langle 2, 4\rangle$, but $Din(2) = 1$ and $Din(4) = 3$, i.e., $Din(2) \neq Din(4) - 1$.

Moreover, it is evident that if arc $\langle u, v\rangle$ exists in the tree, then all v descendants, as well as v itself, are descendants of u. This fact is expressed by the constraints of the form $\sum_{u \in V_{i+1}} (Dout(u) + 1) \times x_{v,u} = Dout(v)$. The condition that every tree node has only one parent is captured by the constraints of the form $\sum_{u \in V_{i-1}} x_{u,v} = 1$.

We should notice that the value of every candidate edge is integral in every *LP2* instance solution (for a proof see [23]):

Theorem 2. *Every solution,* $\boldsymbol{x^*}$ *of an LP2 instance is integral. Specifically, it holds that* $\boldsymbol{x^*} \in \{0, 1\}^n$.

Directed Acyclic Graphs. We can reduce the problem of generating a TC DAG given $Dout$ and Din^- as follows:

(LP3). Let G^* be a transitively closed DAG and E^* be the set of its candidate arcs. Then, the generation of G^* given its out-degree sequence, $Dout$ and a sequence Din^-, s.t., $Din^-(u)$ denotes the level of node u in G^*, can be reduced to an LP instance of the form:

$$\begin{aligned} min\ \mathbf{0}^T\boldsymbol{x} & \\ \textstyle\sum_{(u,v)\in E^*} x_{u,v} = Dout(u), & \quad \forall u \in V \\ \textstyle\sum_{(v,u)\in E^*} x_{v,u} \geq Din^-(u), & \quad \forall u \in V \\ x_{u,v} + x_{v,w} - x_{u,w} \leq 1, & \quad \forall \langle u, v\rangle,\ \langle v, w\rangle \in E^* \\ 0 \leq x_{u,v} \leq 1, & \quad \forall \langle u, v\rangle \in E^* \end{aligned}$$

To determine the candidate arcs of G, we observe that, an arc $\langle u, v\rangle \in V \times V$ can be an arc of a transitively closed DAG $G^* : (V, E^*)$ only if, $Dout(u) > Dout(v)$ and $Din(u) < Din(v)$). We should notice that the strict inequalities guarantee

that no cyclic path can be obtained by the considered set of candidate arcs[3]. Otherwise, all the nodes of the cyclic path would have exactly the same out-and in-degree, since G^* is transitively closed.

The constraint $x_{u,v} + x_{v,w} - x_{u,w} \leq 1$ guarantees that whenever both arcs $x_{u,v}, x_{v,w}$ exist (i.e., $x_{u,v} = x_{v,w} = 1$), the transitive arc $x_{u,w}$ also exists. This is justified as follows: whenever either $x_{u,v}$ or $x_{v,w}$ does not exist, the above inequality obviously holds, since it is of the form $a - b \leq 1$, where $a, b \in [0, 1]$. We focus on the case that both $x_{u,v}$ and $x_{v,w}$ exist. Then, $x_{u,w}$ also exists because G^* is transitively closed. Hence, $x_{u,v} + x_{v,w} - x_{u,w} = 1 + 1 - 1 \leq 1$, which obviously holds as equality.

We should mention that (unlike *LP1* and *LP2*) the solutions of *LP3* instances, are not integral in the general case. As a consequence, we should devise a method for obtaining the set of arcs of the generated graph. To this end, we can consider a threshold value $T \in [0, 1]$ and that an arc $\langle u, v \rangle$ exists iff $x_{u,v} \geq T$. In Section 4 we will examine threshold values that yield good approximations with respect to the given `power-law`.

Handling Infeasibility of LP instances. If the produced *LP2*/*LP3* instance is infeasible, we conclude that there does not exist a tree/DAG whose transitive closure simultaneously realizes the given sequences. In that case we go back to step (1). *PDFSampling* uses a pseudo-random number generator and consequently a different *Dout* is produced for each *PDFSampling* method call.

3.4 Combining Both Graphs

At the last PoweRGen step we consider that $G_s : (V_s, E_s)$ and $G_p : (V_p, E_p)$ are generated. Since they have the same set of nodes we should define an one-to-one function $h : V_s \rightarrow V_p$ that maps each node of $\mathcal{G}_s$ to one and only node of $\mathcal{G}_p$. To this end, we exploit the fact that nodes with high out-degree in the $\mathcal{G}_p$ are located highly in the $\mathcal{G}_s$. Specifically, let k be the level of $\mathcal{G}_s^*$ at which the source nodes (corresponding to domains of properties) of most arcs of $\mathcal{G}_p$ are located. We order the nodes of V_p in descending out-degree order and we reach a list P. Also, let V_i be the set of nodes of $\mathcal{G}_s^*$ located at level i, i.e., $V_i = \{v \in V_s \mid Din(v) = i\}$. Then, we map each of the first $|V_k|$ nodes of P to one and only node of V_k. Similarly, we map the next $|V_{k+1}|$ nodes of P to one and only node of V_{k+1} and the next $|V_{k-1}|$ nodes of P to one and only node of V_{k-1}. This process continues until we map the nodes of V_0 and of V_d to nodes of V_p.

4 Experimental Evaluation

In this Section we experimentally evaluate PoweRGen on two axis, namely, the effectiveness and the efficiency of the proposed algorithm. Figure 4 (resp. Figure 5) shows the generated $\mathcal{G}_p$ (resp. $\mathcal{G}_s$) of a schema with 300 classes, 1000

[3] Replace inequalities with equalities to allow cyclic paths.

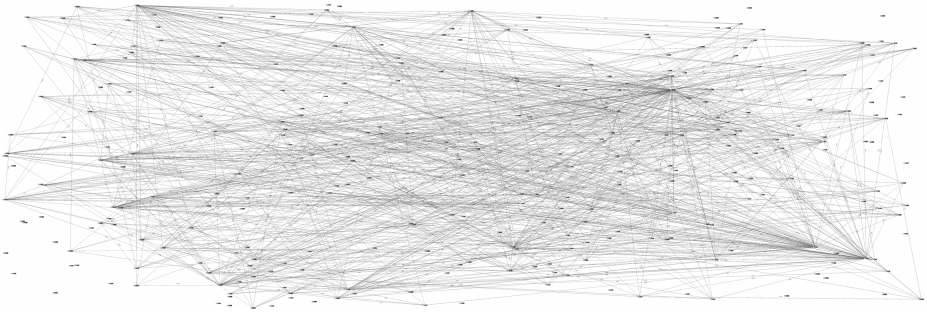


Fig. 4. Synthetic $\mathcal{G}_p$ example

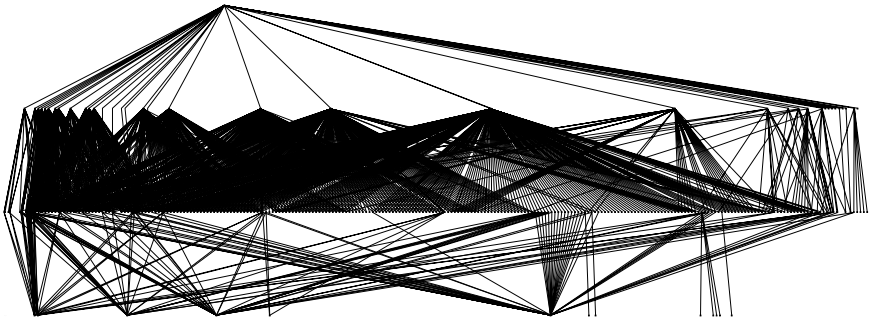


Fig. 5. Synthetic $\mathcal{G}_s$ example

properties, total-degree VR exp. 0.6, $depth = 5$ and class descendants PDF exp. 1.7. The figures have been drawn using *StarLion* [26] and *VRP* [1] visualization facilities.

4.1 Effectiveness

To investigate to what extent a function approximates a `power-law`, we rely on a commonly used method (based on the least square errors method), called Linear Regression [20], to fit a line in a set of 2-dimensional points and, thus, to investigate whether the *log-log* plot of a function approximates a line. The accuracy of the approximation is indicated by the correlation coefficient, the absolute value of which (hereafter called ACC) always lies in $[0, 1]$. An ACC value 1.0 indicates perfect linear correlation, i.e., the points are exactly on a line. Additionally, if PDF follows a `power-law` with exponent β, i.e., $P(X = x) = \alpha x^{-\beta}$, then CCDF follows a `power-law` with exponent $\beta - 1$, i.e., $P(X \geq x) = \gamma x^{-(\beta-1)}$ (see [7] for details). Based on this fact, we do *Linear Regression* on CCDF plots [7] and the results regarding PDF immediately follow.

Total degree CCDF and VR functions of $\mathcal{G}_p$ are illustrated in Figure 6 (upper). As one can observe the CCDF is almost a `power-law` ($ACC = 0.98$). This small

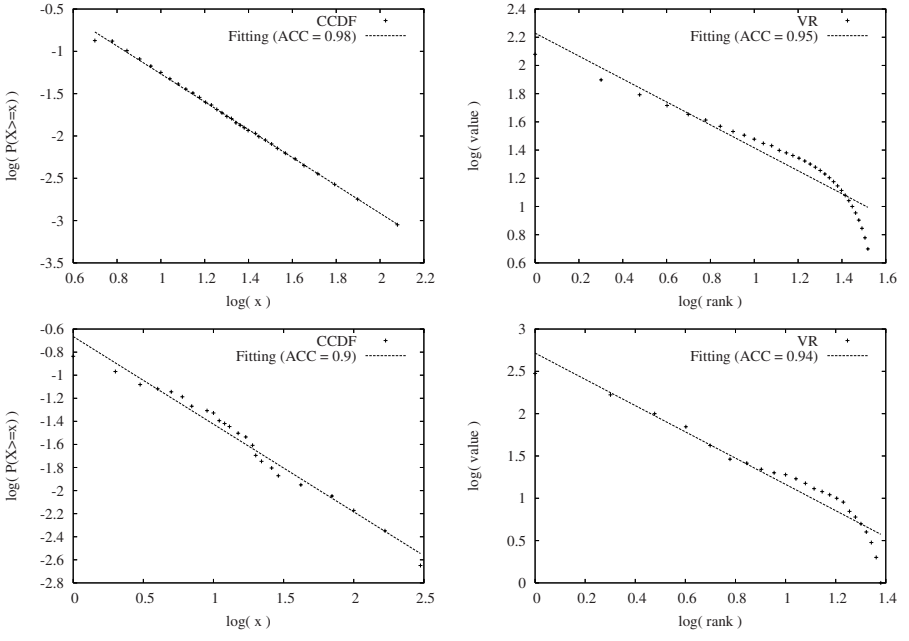


Fig. 6. $\mathcal{G}_p$ total-degree and $\mathcal{G}_s^*$ out-degree functions of our example

divergence from a strict `power-law` (i.e., $ACC = 1.0$) is due to the *VRSampling* method (see Section 2.1).

In the case of $\mathcal{G}_p$ and trees for $\mathcal{G}_s$, we generate schemas whose degree sequences are sampled by the `power-law` distributions assumed as input. On the other hand, in the case of DAGs for $\mathcal{G}_s$ the *LP3* instance solutions usually have coordinates (corresponding to arcs of $\mathcal{G}_s^*$) that are real numbers in $(0, 1)$. To reach a final set of arcs we will consider a threshold value $T \in [0, 1]$: an arc $\langle u, v\rangle$ exists iff $x_{u,v} \geq T$. The use of such heuristic may miss some transitive arcs of $\mathcal{G}_s^*$. For instance, consider that $T = 0.6$ and in the LP solution $x_{u,v} = 0.7$, $x_{v,w} = 0.8$ and $x_{u,w} = 0.5$. Then, $x_{u,v} + x_{v,w} - x_{u,w} = 1 \leq 1$, as required for the arcs of a TC DAG. However, to make the solution integral, we consider that $x_{u,v} = x_{v,w} = 1$ and $x_{u,w} = 0$. Concerning the value of T, we should notice that a big threshold value (e.g., $T = 0.9$), decreases the probability of missing transitive arcs, but ignores many arcs and thus the generated *Dout* and *Din* roughly diverge from the given ones. On the other hand, a small threshold value (e.g., $T = 0.1$), results in bigger number of arcs than in the given sequences. After experiments we found that $T = 0.6$ leads to best approximations with respect to the given `power-law`.

To measure the quality of the approximations yielded by the above threshold, we generated 1000 subsumption DAGs with parameters: 300 classes, $depth = 5$ and $b = 1.7$. Table 1 shows the distribution of the exponents and the ACC values of the generated subsumption graphs. The average computed value for b is

Table 1. PDF `power-law` exponent and ACC of 1000 subsumption graphs with given $b = 1.7$

	Min	**Max**	**Mean**	**St.dev.**	**COV**
b	1.6	1.767	1.691	0.032	0.047
ACC	0.874	0.922	0.903	0.006	0.007

1.691, while the average ACC value is 0.903. We conclude that the out-degree distribution of $\mathcal{G}_s^*$ approximates ($ACC = 0.903$, i.e., 90.3%) a `power-law` whose characteristic PDF exponent approximates ($\frac{1.691}{1.7} = 0.994$, i.e., 99.4%) 1.7. Figure 6 (bottom) illustrates the CCDF and the VR functions of the out-degree of one of the 1000 generated subsumption graphs.

4.2 Efficiency

In this Section we report the time and memory requirements of the LP instances produced for the generation of synthetic SW schemas of various size. To solve the LP instances we used the academic software Soplex [29]. The measurements regarding memory and runtime reported below are strongly depended on the efficiency of Soplex. Improved measurements can be obtained by using more sophisticated *simplex* implementations, such as the commercial CPLEX [16]. We also measure the number of variables and constraints of the produced LP instances that show the complexity of the reduction of our problem to the LP instances and are independent of the specific software implementation of *simplex*. Experiments were carried out on a PC with a Pentium IV 3.2GHz processor and 2 GB of main memory, over Suse Linux (v10.1).

Figure 7 (upper left) shows the number of variables and constraints of the LP1 instances that are produced to generate $\mathcal{G}_p$ for various number of classes, namely $300-1000$. Specifically, we considered as constant $b = 0.6$ and $N_p = 1000$ and we vary N_c. Figure 7 (upper right) shows the time and space requirements of Soplex to solve the *LP1* instances produced to generate $\mathcal{G}_p$. As one can observe, the generation of typically sized schemas with 700 classes and 1000 properties [25] needs 60 sec and 114 MB memory.

Figure 7 (middle left) shows the number of variables and constraints of the *LP3* instances that are produced to generate DAGs for $\mathcal{G}_s$ for various number of classes, namely $100-700$. Specifically, we considered as constant $b=2.2$, $depth=7$ and vary N_c. Figure 7 (middle right) shows the time and space requirements of Soplex to solve the *LP3* instances. We should mention that Soplex crashes for schemas with more than 700 classes. This is due to the transitivity constraints, whose number equals the number of all triples u, v, w of nodes, such that $\langle u, v\rangle$, $\langle v, w\rangle$ are candidate arcs.

On the other hand, in case of trees, the generation of $\mathcal{G}_s$ is much more efficient. To show this fact we considered the degree sequences of the *transitive closures* of trees whose 75% of nodes are leaves and we reduce their generation to an *LP2* instance. We exponentially increase the number of classes, which is computed as a

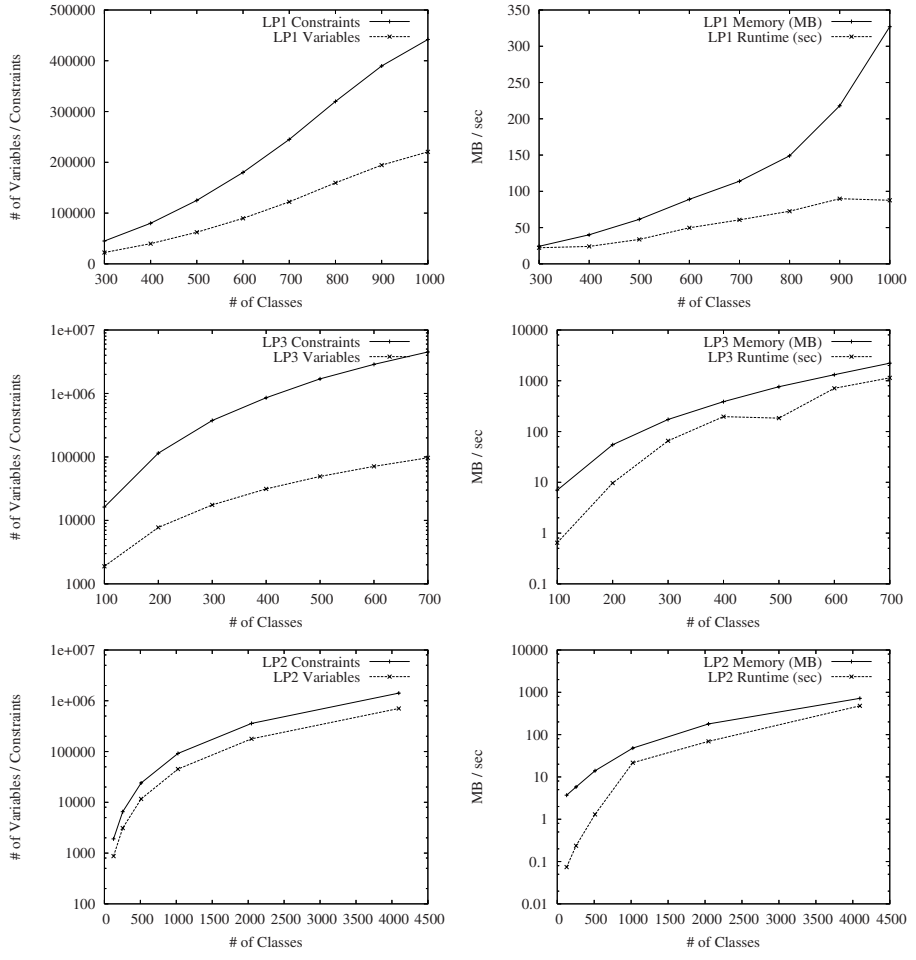


Fig. 7. Efficiency measures for *LP1* (upper) / *LP3* (middle) / *LP2* (bottom) instances

power of 2. Figure 7 (bottom left) shows the number of variables and constraints of the produced *LP2* instances, while Figure 7 (bottom right) the corresponding runtime and memory requirements.

4.3 Experimental Conclusions

As a general remark, PoweRGen produces synthetic property and subsumption graphs whose distributions respect the `power-law` exponents given as input with a confidence ranging between $90 - 98\%$. Regarding efficiency, PoweRGen is scalable concerning the generation of G_p and of G_s in case of trees. On the other hand, in the case of DAGs for G_s, the memory needs rapidly increase with

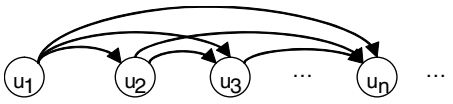


Fig. 8. Algorithm [12,13] Generating Undirected Graphs

respect to the number of classes. However, we should notice, that, even in that case, we can generate schemas up to 700 classes, which exceeds the class number of most real RDFS schemas [23].

5 Related Work

To the best of our knowledge, there does not exist any related work on a general purpose synthetic RDFS schema generator. A recent work [22] has proposed a method to generate small sized schemas by varying the URIs of resources. The motivation of that generator was to evaluate ontology mapping methods. Therefore the focus on the generation process was given to the way that URIs are produced, e.g. a URI is substring of an other. On the other hand, PoweRGen imitates graph features RDFS schemas frequently exhibit in reality and the URIs of the produced properties are of very minor interest, e.g. we assign typical URIs of the form Prop1, Prop2 etc. [11,19] propose a simplistic method to generate SW data as instances of an RDFS (or OWL) schema given as input.

Moreover, [21,2] focus on the generation of XML data that are valid against either a predefined DTD [21] or an XML schema given by the user [2]. However, the full potential of the SW lies on the existence of schemas, which can be exploited to support advanced reasoning services against the available SW data. As a consequence, we need to also generate the schemas, except for the data. Note also, that SW schemas (as well as instance descriptions) are actually graphs, while XML documents are trees.

In addition, [12,13] considered only the generation of simple graphs (undirected, without self-loops and multiple-arcs). Their work is based on the result of [8,10], which can be summarized as follows. Let n denote the number of nodes of the graph we wish to generate. Let u_i, $1 \leq i \leq n$ denote nodes and $d_1 \geq d_2 \geq ... \geq d_n$ intended degrees of these nodes. The necessary and sufficient condition for a degree sequence to be realizable is: $\sum_{i=1}^{k} d_i \leq k(k-1)+\sum_{i=k+1}^{n} min\{k, d_i\}$.

The algorithm is iterative and maintains the residual degrees of vertices. In each iteration step it picks an arbitrary vertex u and adds arcs from u to d_u vertices of highest residual degree, where d_u is the residual degree of u (see Figure 8). The residual degrees of the latter d_u vertices are updated appropriately. By connecting with d_u highest degree vertices the algorithm ensures that the necessary and sufficient condition holds for the residual problem instance. This algorithm is not suitable for synthetic SW schema generation. Specifically, for the generation of G_s we need to consider the arcs transitivity. Moreover, both G_s and G_p are directed graphs.

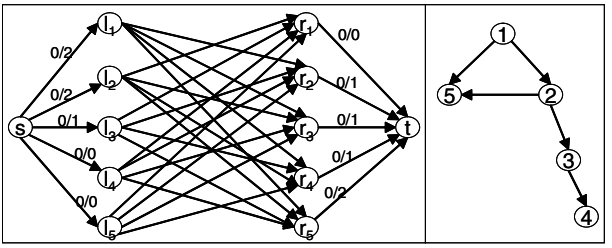


Fig. 9. Algorithm [18] Generating Directed Graphs (left) and its output (right)

Moreover, [18] proposed a reduction of the problem of generating *directed* graphs that simultaneously satisfy two given degree sequences to the *Max Flow* problem. In particular, let $d_{in} = (d_{in,1}, d_{in,2}, ..., d_{in,n})$ and $d_{out} = (d_{out,1}, d_{out,2}, ..., d_{out,n})$ be sequences of integers (in no particular sorted order), with $\sum_{i=1}^{n} d_{in,i} = \sum_{i=1}^{n} d_{out,i}$. We wish to construct a directed graph on n nodes, such that node u_i has $d_{in,i}$ incoming arcs and $d_{out,i}$ outgoing arcs, $1 \leq i \leq n$. Consider the graph that has a source s, a sink t, a set of nodes $L = \{l_1, ..., l_n\}$ and a set of nodes $R = \{r_1, ..., r_n\}$. There is a link of capacity 1 directed from each l_i to each r_j, for $1 \leq i, j \leq n$ and $i \neq j$. There is a link of capacity $d_{out,i}$ directed from s to each l_i, for $1 \leq i, j \leq n$. Finally, there is a link of capacity $d_i n, i$ directed from each r_i to t, for $1 \leq i, j \leq n$. We may now consider integral maximum flows from s to t. If there is such a flow of value $\sum_{i=1}^{n} d_{in,i} = \sum_{i=1}^{n} d_{out,i}$, then the corresponding degree sequences are simultaneously realizable and the flow gives a directed graph that satisfies, simultaneously, in-degrees $d_{in,i}$ and out-degrees $d_{out,i}$.

Figure 9 illustrates an example for this algorithm, where $d_{in} = \langle 0, 1, 1, 1, 2 \rangle$ and $d_{out} = \langle 2, 2, 1, 0, 0 \rangle$. The maximum integral flow of the graph of Figure 9 (left), is $maxFlow = 5 = \sum_{d \in d_{in}} d = \sum_{d \in d_{out}} d$. Hence, d_{in}, d_{out} are simultaneously realizable and the graph of the maximum flow, which is drawn in Figure 9 (right), satisfies them. This algorithm can be used for the generation of G_p, but not of that of G_s since: a) the arcs transitivity needs to be considered and b) G_s is a DAG (i.e., no cycles are allowed).

Furthermore, [5] studied the number of *simple undirected graphs* that realize a given sequence and proposed non deterministic algorithms to generate one of them. Specifically, they focused on how each one of the graphs that realize the given sequence is generated from their algorithms with as uniform distribution as possible. Although theoretically interesting, that work cannot be used in our case, given that it considers undirected graphs (while both $\mathcal{G}_p$ and $\mathcal{G}_s$ are directed) and also ignores the transitivity of arcs (needed to be considered for $\mathcal{G}_s$).

Finally, there exist works that generate graphs exhibiting `power-law` degree distributions (see [7] for a survey). However, a) the `power-law` arise after a big number of node insertions in the graph, b) the characteristic exponent is fixed or is constrained to big values (e.g., bigger than 2) and c) the transitivity of arcs is

not handled. Hence, we can not use these generators, because a) RDFS schema graphs are relatively small sized, b) the `power-law` exponent should be fully parameterizable (it lies in $[0.79, 2.18]$ for total-degree VR, while in $[1.54, 2.47]$ for class descendants PDF for real schemas) and c) we should enforce as much as possible arcs transitivity for $\mathcal{G}_s$.

6 Future Work

As future work, we plan to explore the possibility of devising a more efficient method to generate $\mathcal{G}_s$ when it is considered to be a DAG. To this end, it would be interesting to investigate if graph generation methods exploiting the ideas of *self-similarity* and *fractals* [17], could be modified to take as input the degree sequences, or at least the characteristic `power-law` exponent of their distribution. We also plan to exploit the algorithms presented in this paper for synthetic SW data generation as a part of our ongoing *RDF/S* benchmarking efforts [24].

References

1. Alexaki, S., Christophides, V., Karvounarakis, G., Plexousakis, D., Tolle, K.: The ICS-FORTH RDFSuite: Managing Voluminous RDF Description Bases. In: 2nd International Workshop on the Semantic Web (May 2001)
2. Barbosa, D., Mendelzon, A., Keenleyside, J., Lyons, K.: ToXgene: A template-based data generator for XML. In: Proceedings of the ACM SIGMOD International Conference on Management of Data, New York, p. 616 (2002)
3. Barbosa, D., Mignet, L., Veltri, P.: Studying the XML Web: Gathering Statistics from an XML Sample. World Wide Web Journal 8(4) (2005)
4. Berners-Lee, T., Hendler, J., Lassila, O.: The Semantic Web. Scientific American (May 2001)
5. Blitzstein, J., Diaconis, P.: A Sequential Importance Sampling Algorithm For Generating Random Graphs With Prescribed Degrees. Annals of Applied Probabilty (2005)
6. Brickley, D., Guha, R.V.: RDF Vocabulary Description Language 1.0: RDF Schema, W3C Recommendation (February 10, 2004)
7. Chakrabarti, D., Faloutsos, C.: Graph Mining: Laws, Generators, and Algorithms. ACM Computing Surveys (CSUR) 38(2) (2006)
8. Claude, B.: Graphs and Hypergraphs. North Holland Publishing Company, Amsterdam (1973)
9. Wang, T.D.: Gauging Ontologies and Schemas by Numbers. In: Proc. Fourth Int'l Workshop Evaluation of Ontology-Based Tools (EON) (2006)
10. Erdös, P., Gallai, T.: Graphs with Prescribed Degree of Vertices. Mat. Lapok 11, 264–274 (1960)
11. Guo, Y., Heflin, J., Pan, Z.: Benchmarking DAML+OIL Repositories. In: Fensel, D., Sycara, K.P., Mylopoulos, J. (eds.) ISWC 2003. LNCS, vol. 2870, pp. 613–627. Springer, Heidelberg (2003)
12. Hakimi, S.: On the Realizability of a Set of Integers as Degrees of the Vertices of a Graph. SIAM 10, 496–506 (1962)

13. Havel, V.: A Remark on the Existence of Finite Graphs. Casopis Pest. Mat. 80, 477–480 (1955)
14. Hayes, P.: RDF Semantics. W3C Recommendation (February 10, 2004)
15. Hoffman, A.J., Kruskal, J.B.: Integral Bounding Points of Convex Polyedra. In: Linear Inequalities and Related Systems, pp. 223–246. Princeton University Press, Princeton (1956)
16. Hsu, M., Cheatham, T.E.: Rule Execution in CPLEX: A Persistent Objectbase. In: Dittrich, K.R. (ed.) OODBS 1988. LNCS, vol. 334, pp. 150–155. Springer, Heidelberg (1988)
17. Leskovec, J., Chakrabarti, D., Kleinberg, J.M., Faloutsos, C.: Realistic, Mathematically Tractable Graph Generation and Evolution, Using Kronecker Multiplication. In: Jorge, A.M., Torgo, L., Brazdil, P.B., Camacho, R., Gama, J. (eds.) PKDD 2005. LNCS (LNAI), vol. 3721, pp. 133–145. Springer, Heidelberg (2005)
18. Mihail, M., Visnoi, N.: On Generating Graphs with Prescribed Degree Sequences for Complex Network Modeling Applications. In: Procs of Approx. and Randomized Algorithms for Communication Networks (ARACNE) (2002)
19. Perry, M.: Test ontology generation tool, `http://lsdis.cs.uga.edu/projects/semdis/tontogen/`
20. Press, W.H., Teukolsky, S.A., Vetterling, W.T., Flannery, B.P.: Numerical Recipes in C, 2nd edn. Cambridge University Press, Cambridge (1992)
21. Schmidt, A., Waas, F., Kersten, M., Carey, M.J., Manolescu, I., Busse, R.: Xmark: A benchmark for XML data management. In: Procs of the 28th International Conference on Very Large Data Bases, Hong Kong, China, pp. 974–985 (2002)
22. Svab, O., Svatek, V.: Vitro Study of Mapping Method Interactions in a Name Pattern Landscape. In: 2nd International Workshop on Ontology Matching, collocated with ISWC 2007 (2007)
23. Theoharis, Y.: On Power Laws and the Semantic Web. Master's thesis, Computer Science Department, University of Crete (February 2007), `http://athena.ics.forth.gr:9090/RDF/publications/MasterThesisTheohari.pdf`
24. Theoharis, Y., Christophides, V., Karvounarakis, G.: Benchmarking Database Representations of RDF/S Stores. In: Gil, Y., Motta, E., Benjamins, V.R., Musen, M.A. (eds.) ISWC 2005. LNCS, vol. 3729, pp. 685–701. Springer, Heidelberg (2005)
25. Theoharis, Y., Tzitzikas, Y., Kotzinos, D., Christophides, V.: On Graph Features of Semantic Web Schemas. IEEE Transactions on Knowledge and Data Engineering 20(5), 692–702 (2008)
26. Tzitzikas, Y., Kotzinos, D., Theoharis, Y.: On Ranking RDF Schema Elements (and its Application in Visualization). Journal of Universal Computer Science, Special Issue: Ontologies and their Application 13(12), 1854–1880 (2007)
27. Veinott, A.F., Dantzig, G.B.: Integral Extreme Points. SIAM Review 10(3), 371–372 (1968)
28. W3C. W3C Semantic Web Activity, W3C Workshop on RDF Access to Relational Databases (October 25-26, 2007)
29. Wunderling, R.: Paralleler und Objektorientierter Simplex-Algorithmus. Ph.D. thesis, ZIB (1996)