Actes JIAF-JFPDA 2023

A New Evolutive Generator for Graphs with Communities and its Application to Abstract Argumentation

Jean-Marie Lagniez[1] **Emmanuel Lonca**[1] **Jean-Guy Mailly**[2] **Julien Rossit**[2]
[1] CRIL, Univ. Artois, CNRS, France
[2] Université Paris Cité, LIPADE, F-75006 Paris, France
{lagniez, lonca}@cril.fr
{jean-guy.mailly, julien.rossit}@u-paris.fr

Abstract

Graph generators are a powerful tool to provide benchmarks for various subfields of KR (e.g. abstract argumentation, description logics, etc.) as well as other domains of AI (e.g. resources allocation, gossip problem, etc.). In this paper, we describe a new approach for generating graphs based on the idea of communities, i.e. parts of the graph which are densely connected, but with fewer connections between different communities. We discuss the design of an application named crusti_g2io implementing this idea, and then focus on a use case related to abstract argumentation. We show how crusti_g2io can be used to generate structured hard argumentation instances which are challenging for the fourth International Competition on Computational Models of Argumentation (ICCMA'21) solvers.

1 Introduction

Graph-based models are widespread in many fields of Knowledge Representation and Reasoning [10] (e.g. abstract argumentation [12], description logics [28], etc.) as well as other domains of Artificial Intelligence like multi-agent systems (e.g. resources allocation [4], gossip problem [11], etc.).

The popularity of this kind of representation appeals automated graphs generation approaches to provide challenging benchmarks that can put to the test practical tools developed within these various frameworks. The literature offers different methods to generate graphs, which exhibit different properties and various applicabilities to concrete problems and scenarios. In particular, one challenge consists in generating *structured* instances, i.e. random graphs which present interesting patterns that are relevant for some specific application. A well-known example of such a structured generation model is the Watts-Strogatz model [31], where the generated graphs have a *small world* property. Among the variety of graphs that have been studied, some recent works are interested in the generation of graphs with communities of nodes, i.e. parts of the graphs which are densely connected, but with fewer connections between different communities [20]. Such models include BTER [23] and Darwini [17], that propose to link nodes inside so-called affinity blocks, and then to add links between the nodes from different blocks. This kind of graphs is for example able to model interactions between people, including in social networks [26]. The importance of generators for this kind of graphs is amplified by the privacy issues that come when using real social networks data [32].

Being a model of choice to represent people communities, graphs with communities are a *de facto* candidate to encode large debates, which could be the source of argumentative reasoning. Computational argumentation has become an important sub-field of Knowledge Representation and Reasoning, being a prominent formalism for non-monotonic reasoning [12] in general, and reasoning with inconsistent knowledge in particular [3].

However, until recently, there was an important lack of practical approach for computing the solutions of argumentation problems. Although there were some algorithmic approaches proposed in the literature, few pieces of software were actually available for the community. This has changed (mainly) thanks to the organization of the First International Competition on Computational Models of Argumentation (ICCMA), in 2015. Since then, some solvers have been proposed, based either on original techniques dedicated to argumentation frameworks [19, 21, 22], or on translation into other frameworks which have already proven efficient computational benefits (namely Boolean satisfaction problem (SAT) [16, 24, 29], Answer Set Programming (ASP) [15] or Constraint Satisfaction Problems (CSP) [5]). The efforts of the community at the occasion of the various editions of ICCMA have seen a general increase of the quality of the computational approaches for argumentation, both with respect to the correctness of the approaches and their runtime efficiency. However, the lack of challenging and realistic benchmarks for argumentation is still an issue for the community. Using (community-based) graph generators was naturally quickly considered to fill this hole.

BTER and Darwini approches are customizable in the sense that some metrics can be given to produce graphs with communities of expected shapes, but the manner the communities are linked is tied with the community generation algorithm which follows the Erdös-Rényi model [18]. In this paper, we propose a new generation approach and we apply it to abstract argumentation.

Our approach is based on three components : we first generate an *outer graph* which gives a global skeleton for the structure of the generated instance ; then in each node of the outer graph, we generate an *inner graph* i.e. a community of nodes ; and finally when two nodes of the outer graph are connected, we use a *linker* to add some relations between the corresponding inner graphs. We then show how our method can be applied to generate structured, challenging graphs for argumentation purpose. The added value of our approach compared to the previous ones lies in its ability to be generic and modular, since any of the three components can be easily replaced by other versions. In particular, the outer and inner graphs can be generated through classical generation models like Erdös-Rényi [18], Watts-Strogatz [31] or Barabási-Albert [1], but any other model could be plugged instead (including BTER and Darwini graphs themselves). Our contribution includes a documented, open-source graph generator following this inner/outer template. This application has been made to be easily used by any user, but also to be convenient for developers who want to add new features like graph generators, linkers or output formats.

The paper is organized as follows. After providing some necessary background in Section 2, we first introduce the inner/outer model in Section 3. This model is then instantiated to generate abstract argumentation benchmarks in Section 4. Section 5 presents some related works. Necessary and relevant features of our framework are presented in Section 6, followed by some experiments in Section 7. Finally, Section 8 draws some conclusions and highlights avenues for future work.

2 Background on Graph Generators

Let us first describe various classical graph generation models, which are later used in the conception of our new approach. In the following, we use $G = \langle N, E\rangle$ to denote any graph, where N are the nodes and E are the edges. In the case of a directed graph, $E \subseteq N \times N$, while in the case of a non-directed graph, $E \subseteq \{\{a, a'\} \mid a, a' \in N\}$. We also consider simple models like *paths* and *trees*.

Erdös-Rényi The Erdöz-Rényi (or binomial graph) generation model [18] takes into consideration two parameters $n_e \in \mathbb{N}$ and $p_e \in]0, 1]$ to construct graphs $\langle N, E\rangle$ with $|N| = n_e$ nodes, where for each couple $(a_i, a_j) \in N \times N$ there is a probability p_e to add an edge (a_i, a_j) in E.

Watts-Strogatz The model proposed in [31] considers a number of arguments $n_w \in \mathbb{N}$ and an even number $k_w \in \mathbb{N}$ (s.t. $k_w < n_w$) to construct a ring lattice made of n_w nodes, where each node is linked to k_w other nodes. Then, for each node a and each edge (a, b) of this node, there is a probability p_w of re-wiring the edge (avoiding to duplicate an existing edge or to link the node a with itself). Such graphs are called *small worlds*, i.e. for any two nodes in the graph, the shortest path between them has a logarithmic length in the number of nodes.

Barabási-Albert The preferential attachment model by [1] is based on two parameters $n_b, m_b \in \mathbb{N}$. It allows to generate graphs $\langle N, E\rangle$ where $|N| = n_b$, which are built by incrementally enlarging an initial graph (possibly made of a single node), such that each new node is attached to m_b nodes with a preference for existing nodes with the higher degree (formally, the probability to attach a new node a to an existing node b is $p_b = \frac{deg(b)}{\sum_c deg(c)}$ where $deg(b)$ (resp. $deg(c)$) is the degree of b. (resp. of c), and c iterates over the set of nodes already present in the graph).

Community-based Graphs Some models have already been proposed in the literature to incorporate the notion of community within the structure of the graphs, such as BTER and Darwini. BTER [23] splits a set of k nodes into so-called affinity blocks (i.e. the communities of nodes), which are then locally linked, and finally nodes from different blocks are linked together. Affinity blocks are linked following the Erdös-Rényi model, while the links between different blocks use the Chung-Lu model [9] (which is an extension of the Erdös-Rényi model). Darwini [17] performs a similar process, with an additional starting point which consists in mapping each node with its degree and clustering coefficient.

Directed/Undirected Graphs In the definition that we provide for the Erdös-Rényi model, we assume that the graph is directed. It is easy to obtain a non-directed graph by choosing to add an (undirected) edge $\{a_i, a_j\}$ with a probability p_e (instead of considering both the directed edges (a_i, a_j) and (a_j, a_i)). Similarly, obtaining a directed path is easy (once the non-directed graph made of a single path $(a_1, a_2, \ldots, a_n)$ is built, each edge is directed from a_i to a_{i+1}, for each $i \in \{1, \ldots, n-1\}$). In the case of trees, we can also easily build a directed graph, for instance with the edges going "down" from the root to the leaves.

Unfortunately, the graphs generated by the other models are generally non-directed. When a directed graph is required, it could be possible to randomly select the orientation of each edges. However, depending the targeted application, this solution is still not satisfactory. For example, when considering the problem of generating argumentation frameworks, it is important to consider symmetrical attacks between argument in order to cover a wide range of cases. To do so, an option consists in considering a parameter $p_s \in [0, 1]$ representing the probability that a given edge should be symmetrical. Then, for an edge $\{a_i, a_j\}$ in the non-directed graph, there will be a probability p_s to have both (a_i, a_j) and (a_j, a_i) in the directed version of the graph, and a probability $\frac{1-p_s}{2}$ for either (a_i, a_j) or (a_j, a_i).

3 The Inner/outer Model

As mentioned earlier, existing community-based graphs generators suffer from being tied to the model used to build their communities. In order to overcome this issue, we propose a new approach for generating graphs that considers underlying graph structures. Roughly speaking, we implement the reverse approach of the BTER process : we first generate the relations between the communities, then we generate communities and finally we link them by connecting some of their inner elements. More precisely, an *outer graph* $G_{\mathcal{G}^O}$ that will be used as a skeleton for the instance is first constructed from a graph generator $\mathcal{G}^O$. Then, each node of this graph is associated with a fresh *inner graph* (fresh in the sense where nodes of each inner graph are disjoint) built by another generator $\mathcal{G}^I$. In order to link inner graphs together, we successively consider each inner graph G_n rooted to a node n of $G_{\mathcal{G}^O}$ and add edges between it and the inner graphs $G_{n'}$ rooted to a node n' when an edge exists in the outer graph between n and n'. The final graph is then the set of inner graphs together with the added edges. Interestingly, such generation process can handle both directed and undirected graphs (with the constraint that both generators and the added edges involve edges of the same kind). Formally, the function in charge of linking inner graphs together in the directed case is defined as follows :

Definition 1 (Directed linker) *A* linker over directed graphs *is a mapping $\mathcal{L}_d$ such that, for any $G_1 = \langle N_1, E_1 \rangle$ and $G_2 = \langle N_2, E_2 \rangle$: $\mathcal{L}_d(G_1, G_2) \subseteq (N_1 \times N_2) \cup (N_2 \times N_1)$.*

For the undirected case the linker is defined as follows :

Definition 2 (Undirected linker) *A* linker over undirected graphs *is a mapping $\mathcal{L}_u$ such that, for any $G_1 = \langle N_1, E_1 \rangle$ and $G_2 = \langle N_2, E_2 \rangle$: $\mathcal{L}_u(G_1, G_2) \subseteq \{\{n_1, n_2\} \mid n_1 \in N_1, n_2 \in N_2\}$.*

Without loss of generality, in the following we only consider the directed case. Algorithm 1 formalizes our approach.

Algorithm 1 Inner/outer graph generation

Input: an outer graph generator $\mathcal{G}^O$, an inner graph generator $\mathcal{G}^I$ and a linker $\mathcal{L}$
Output: an inner/outer graph
1: $G_{\mathcal{G}^O} \leftarrow \langle N, E \rangle$ a $\mathcal{G}^O$-generated graph
2: **for** $n \in N$ **do**
3: $\quad G_n \leftarrow \langle N_n, E_n \rangle$ a $\mathcal{G}^I$-generated graph
4: **end for**
5: $L = \emptyset$
6: **for** $(n, n') \in E$ **do**
7: $\quad L \leftarrow L \cup \mathcal{L}(G_n, G_{n'})$
8: **end for**
9: **return** $\langle (\bigcup_{n \in N} N_n), (\bigcup_{n \in N} E_n) \cup L \rangle$

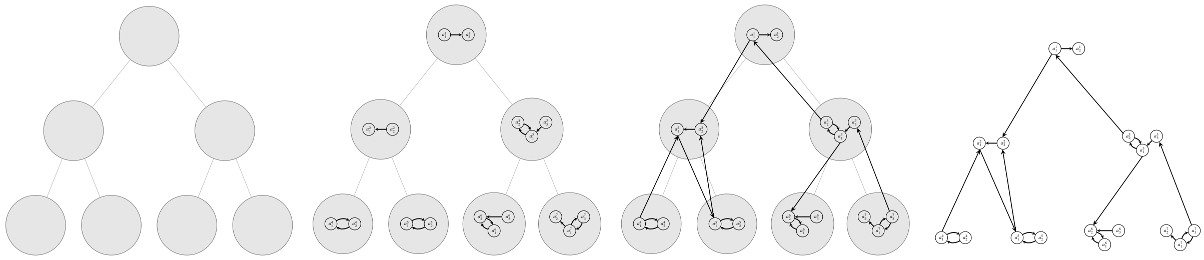
The generation process starts with the generation of the outer graph, i.e. the graph which is used as the skeleton of the instance (line 1). Then, each node of this outer graph is associated with an inner graph which is built by the dedicated graph generator $\mathcal{G}^I$ (line 3). The rest of the algorithm consists in building some links between the different inner graphs, with respect to the structure of the outer graph. To do so, for each edge in the outer graph, the inner graphs associated with the two outer graph nodes under consideration are passed to the linker (line 7) ; the resulting set of edges is stored. At the end, the algorithm returns the union of the inner graphs plus the edges returned by the linker, producing the final inner/outer graph.

Our approach offers the advantage of being flexible and allows, for instance, to generate a community graph such that the outer graph is a tree ($\mathcal{T}$) and inner graphs are Erdös-Rényi graphs ($\mathcal{ER}$). It is also possible to generate paths of Barabási-Albert ($\mathcal{BA}$) graphs, or Watts-Strogatz ($\mathcal{WS}$) graphs made of $\mathcal{WS}$ communities, etc.

Example 1 *Let us illustrate the generation algorithm with $\mathcal{G}^O = \mathcal{T}$, $\mathcal{G}^I = \mathcal{ER}$, and $\mathcal{L}$ a function which returns a random set of edges between two graphs. An example of generation process is given at Figure 1. Figure 1a shows the outer graph, which is thus a balanced binary tree. Then, in each node of the tree, an inner graph is generated thanks to the Erdös-Rényi model (Figure 1b). Figure 1c shows the addition of edges between the inner graphs thanks to the linker. And finally, the resulting graph is shown at Figure 1d.*

4 Application to Abstract Argumentation

From a practical point of view, it seems reasonable to assume that large debates may be structured in smaller



(a) Outer Graph (b) Inner Graphs (c) Edges between Inner Graphs (d) Final Graph

FIGURE 1 – Generation process.

sub-debates, which are only connected by few links ; this would follow how people are themselves structured in social networks [26]. More precisely, this can be the case, for instance, in argumentation frameworks related to multi-issue negotiation, where each sub-debate corresponds to the arguments focusing on one issue, and the links between sub-debates correspond e.g. to the concessions ("If I accept to pay more for this car, then I want the company to deliver it faster" makes the link between the sub-debate about the price of the car and the sub-debate about the delivery date). So, in some sense, these sub-debates represent communities of arguments which are strongly related (i.e. there is a high density of attacks in such a community), and there are fewer relations between different communities. In this section, we briefly recall basic notions of abstract argumentation.

Definition 3 *An* abstract argumentation framework *(AF) [12] is a directed graph $\mathcal{F} = \langle A, R \rangle$ where A is a set of* arguments *and $R \subseteq A \times A$ is the* attack relation *between arguments.*

We say that an argument a *attacks* an argument b if $(a, b) \in R$. This is generalized to sets of arguments : S attacks b (resp. S') if there is some $a \in S$ which attacks b (resp. some $b \in S'$). A set S *defends* an argument a if for any b attacking a, there is a $c \in S$ attacking b. Acceptability of arguments is usually evaluated thanks to the notion of extensions, i.e. sets of collectively acceptable arguments. Various semantics exist for defining extension [12]. Formally, a semantics is a function $\sigma : \mathcal{F} = \langle A, R \rangle \mapsto \mathcal{E} \subseteq 2^A$.

Definition 4 *Given an AF $\mathcal{F} = \langle A, R \rangle$, and a set of argument $S \subseteq A$,*

- $S \in \mathtt{cf}(\mathcal{F})$ *iff* $\forall a, b \in S$, $(a, b) \notin R$,
- $S \in \mathtt{ad}(\mathcal{F})$ *iff* $S \in \mathtt{cf}(\mathcal{F})$ *and S defends all its elements,*
- $S \in \mathtt{co}(\mathcal{F})$ *iff* $S \in \mathtt{ad}(\mathcal{F})$ *and S does not defend any argument in* $A \setminus S$,
- $S \in \mathtt{pr}(\mathcal{F})$ *if S is a $\subseteq$-maximal element of* $\mathtt{ad}(\mathcal{F})$,
- $S \in \mathtt{stb}(\mathcal{F})$ *iff* $S \in \mathtt{cf}(\mathcal{F})$ *and S attacks all the arguments in* $A \setminus S$,
- $S \in \mathtt{gr}(\mathcal{F})$ *iff S is the $\subseteq$-minimal element of* $\mathtt{co}(\mathcal{F})$.

where cf, ad, co, pr, stb *and* gr *stand respectively for* conflict-free, admissible, complete, preferred, stable *and* grounded.

See e.g. [12, 2] for more details about these semantics as well as other semantics defined in the literature. Let us illustrate the complete, preferred, stable and grounded semantics with the following example :

Example 2 *The extensions for* co, pr, stb *and* gr *of the AF $\mathcal{F} = \langle A, R \rangle$ depicted in Figure 2 are given in Table 1.*

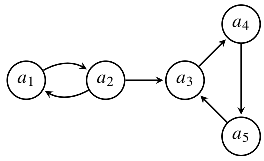


FIGURE 2 – The AF $\mathcal{F}$

Semantics σ	Extensions $\sigma(\mathcal{F})$
co	$\emptyset, \{a_1\}, \{a_2, a_4\}$
pr	$\{a_1\}, \{a_2, a_4\}$
stb	$\{a_2, a_4\}$
gr	$\emptyset$

TABLE 1 – Extensions of the AF $\mathcal{F}$

Recall that reasoning with AFs is generally hard, with many classical problems at the first or second level of the polynomial hierarchy [14].

5 Related Works

The next sections presents the application we developed to generate inner/outer graphs and its application to generate AF benchmarks. There already exists tools for generating AFs from random graph generators. But, from the best of our knowledge, these tools do not modify the underlying graph generated by these models. In [7], the authors propose the C++ framework `AFBenchGen`. It is an AF generator based on the Erdös-Rényi model ($\mathcal{ER}$). In [8], the same authors proposed an extension of `AFBenchGen`, called `AFBenchGen2` which is written in Java, that also consider two additional random graph generator models, which are the Watts-Strogatz ($\mathcal{WS}$) and Barabási-Albert ($\mathcal{BA}$) models. For these two generators the random graphs are used as such. Our tool is much more general than the `AFBenchGen` family of AFs generators. Indeed, by considering the simple graph consisting in one node as outer graph, it is possible to have the exactly same behaviour.

In [25], we introduced a new method for generating challenging benchmarks for the ICCMA'21 competition. This generator is the fundamental basis of our tool. More precisely, we have proposed three variants of our generator $\langle \mathcal{G}^O, \mathcal{G}_i^I, \mathcal{L} \rangle$, with $i \in \{1, 2, 3\}$, defined as follows. In our case $\mathcal{G}^O = \mathcal{T}$, meaning that the underlying graph is actually a perfectly balanced d-tree of height h, where d and h are fixed and provided as parameters. The only difference between the three variants is the inner graphs generator : $\mathcal{G}_1^I = \mathcal{ER}$, $\mathcal{G}_2^I = \mathcal{BA}$, while $\mathcal{G}_3^I$ is a random pick of either $\mathcal{ER}$ or $\mathcal{BA}$, which means that in the first case all the local graphs are Erdös-Rényi graphs, in the second case they are all Barabási-Albert graphs, and in the last case they can be either of them with a probability 0.5.

Once the outer graph has been generated, the inner graphs are linked as follows. For this generation model, the iteration over the set of edges (line 6 in Algorithm 1) is a breadth-first graph traversal from the root to the leaves of the tree. For each inner graph associated with an outer node o, `k` nodes are randomly selected (`k` varies from 5 up to 12 for the benchmarks generated for the ICCMA'21 competition). The descendants $\{o_1, \ldots, o_m\}$ of o are iteratively considered. For each o_i, between 20% and 70% of the inner nodes contained in o_i are randomly selected. Then, for each node n_1 picked in o and with each node n_2 picked in o_i one of the attacks (n_1,n_2) or (n_2,n_1) is added randomly.

In this paper a slightly modified version of the tool proposed for generating the ICCMA'21 benchmarks has been considered. Inner graphs are only linked with their children (and not with any of their descendants). Moreover, a ratio of 20% has been considered for selecting the edges that are added between communities (instead of a ratio between 20% and 70% of the nodes).

6 The crusti_g2io graph generator

We built a command line application called crusti_g2io, dedicated to the generation of inner/outer graphs. It is made available under the terms of the GNU GPLv3 on Github account of the *Centre de Recherche en Informatique de Lens*. [1] We took advantage of the Rust programming language to provide an efficient, memory-safe application, even in parallel context. In addition, Rust allows crusti_g2io to be both an application and a library (the project in mainly a Rust library with additional code to create the application). Interestingly, Rust libraries can be turned into C libraries (static or dynamic) or be linked with them. This makes crusti_g2io able to use any library that can be turned into a C library or to be used itself with any program that can load C libraries, allowing for example Go and Python bindings.

The application can be used to generate both directed and undirected graphs. In the following, we describe how to use the application for directed graphs only ; however, going from directed to undirected is as simple as replacing `directed` by `undirected` in the commands.

The first goal of crusti_g2io is to be easy to install and to use. The only requirement to use it is to have a Rust compiler installed (except of course if you were given an already compiled version) ; then, executing a standard release build command (`cargo build -release`) produces the executable (in the `target/release` directory on UNIX systems). The user can also use the `cargo install` command to compile and install the program on its computer.

From a user perspective, crusti_g2io is made to be used without looking at its documentation. Calling crusti_g2io with `-h`, `-help` displays the list of the commands and what they do. Calling crusti_g2io with a command and one of the two help flags displays the help message associated with the command. For example, calling `crusti_g2io generate-directed -h` explains what `generate-directed` does, gives its mandatory and optional options (along with their descriptions).

The goal of crusti_g2io is to generate a graph from an outer graph generator, an inner graph generator and a linker, and to output it using a graph output format. Thus, these exact four options form the exact set of mandatory options for the `generate-directed` command. Again, they can be recalled by typing `crusti_g2io generate-directed -h` in a terminal. Concerning the lists of the available graph generators, linkers and graph output formats, they can all be retrieved by a crusti_g2io command (respectively `generators-directed`, `linkers-directed` and `display-engines-directed`); calling these commands also indicates how to parameterize the generators, linkers or formats which need it. Figure 3 shows how to build a tree-like outer graph (`-o`) of 10 inner (`-i`) Erdös-Rényi graphs of 100 nodes with a probability of 0.5 where links (`-l`) are created between lowest degree nodes, and export (`-x`) it in the file `t_10_er_100_50.dot` using the dot format (`-f`). The required parameters for generators and linkers (when needed) are given after a slash

1. At the time of submission, it is here : `https://www.cril.univ-artois.fr/~lonca/crusti_g2io-94dfb5e8b6e14a3c13bf9f861b0ad221533815de.zip`.

```
me@PC:~/crusti_g2io generate-directed -o tree/10 -i er/100,0.5 -l min_incoming -x t_10_er_100_50.dot -f dot
![INFO ] [2023-03-03 10:54:39] crusti_g2io 0.1.0
[...]
![INFO ] [2023-03-03 10:54:39] random seed is 6203895736620038422
![INFO ] [2023-03-03 10:54:39] beginning the outer graph generation
![INFO ] [2023-03-03 10:54:39] beginning the inner graphs generation
![INFO ] [2023-03-03 10:54:39] beginning the linking
![INFO ] [2023-03-03 10:54:39] generated a graph with 1000 nodes and 24882 edges
![INFO ] [2023-03-03 10:54:39] exiting successfully after 45.6625ms
```

FIGURE 3 – Example on invocation of crusti_g2io.

and split by commas (see `tree/10` and `er/100,0.5` in the figure). Embedded graph generators include the famous Erdös-Rényi, Watts-Strogatz and Barabási-Albert models, trees and chains. Concerning the linkers, one is a random one, one links nodes with the least incoming edges, and the last one links the nodes with index 0 — which can have some meaning, in particular if a graph is initialized with a special value like in the Barabási-Albert model. Finally, The Graphviz DOT and GraphML formats are available, just like the abstract argumentation related format APX we use in next section.

```rust
#[derive(Default)]
pub struct ErdosRenyiGeneratorFactory;

impl<Ty, R> NamedParam<BoxedGenerator<Ty, R>> for ErdosRenyiGeneratorFactory
where
    R: Rng,
    Ty: EdgeType,
{
    fn name(&self) -> &'static str {
        "er"
    }

    fn description(&self) -> Vec<&'static str> {
        vec![
            "A generator following the Erdős-Rényi model.",
            "First parameter gives the number of nodes of the graph, while t
        ]
    }

    fn expected_parameter_types(&self) -> Vec<ParameterType> {
        vec![ParameterType::PositiveInteger, ParameterType::Probability]
    }

    fn try_with_params(
        &self,
        parameter_values: Vec<ParameterValue>,
    ) -> Result<BoxedGenerator<Ty, R>> {
        let n = parameter_values[0].unwrap_usize();
        let p = parameter_values[1].unwrap_f64();
        Ok(Box::new(move |r| {
            petgraph_gen::random_gnp_graph(r, n, p).into()
        }))
    }
}
```

FIGURE 4 – Implementation of a new graph generator for Erdös-Rényi graphs using the `petgraph` library.

These generators, linkers and formats are a very small subset of what is offered by the literature. This is the reason why we tried to make the addition of new content as easy as possible for developers. For example, to add a new generator, it is only required to create a structure that implements the four functions of the dedicated trait and to register it in the set of generators. Concerning the trait, the implementation of three functions out of four is straightforward (see Figure 4 for an example of implementation for $\mathcal{ER}$ graphs using the `petgraph` library – `https://crates.io/crates/petgraph`), as they respectively return the name of the generator to be used on the command line interface, the description of the generator, and the types of the expected parameters. The last function is the one dedicated to the generation of graphs : it takes as input the (checked) parameter values as given on the command line interface (i.e. the content following the slash) and returns a closure which takes a pseudo-random number generator (PRNG) and produces a graph. The registration of the new generator consist of adding an import statement and a single line of code. Adding a new linker requires a similar process, except that the closure takes a PRNG and two graphs, and returns a vector of edges. When invoking crusti_g2io, the graph can be printed out on the standard output (this is the default behaviour) or exported to a file. The default behavior mixes log messages and the graph ; this can be prevented by hiding the log messages (e.g. by setting the corresponding option) or by exporting the graph to a file. Adding a new output format is similar to adding a new generator or linker.

Finally, crusti_g2io is made to produce reproductible results. By default, it uses an unpredictable random seed ; in order to get reproductible results, the user can set the random seed with the `-s` option on the command line. Regardless of the fact the seed was specified or randomly specified, it is logged so the results can be reproduced. An effort was made in order to mix reproducibility and the use of the full power of the computers, as the application computes the inner graphs and the links between these graphs in a parallel fashion. In order to get reproductible results, the program first computes the outer graph using the global PRNG initialized with the provided seed. Then, each outer node is sequentially associated a random seed using the global PRNG. This way, each inner graph generation process can receive a PRNG which directly depends on the CLI-provided seed, enforcing the reproducibility of the generation for a given seed. The same approach is used for the linking process.

7 Using crusti_g2io to generate challenging abstract argumentation problems

In this section, we use crusti_g2io to generate structured instances for abstract argumentation solvers. The goal is to generate overall challenging instances composed of multiple communities. In addition, we want to generate instances with a large amount of small communities, but also instances with less communities of a greater size. To achieve this, we aim at drawing the frontier between hard and too-hard instances for a set of community sizes, densities and counts.

In order to evaluate the difficulty induced by the generated argumentation graphs, we chose to compute extensions (putting acceptance queries aside) to consider the whole graphs instead of problems that could be related to a reduced area of the graph. We arbitrary selected a problem of the first level of the polynomial hierarchy (SE-ST : compute an extension for the stable semantics) and one of the second level (SE-PR : compute an extension for the preferred semantics). For both tracks, we used the solvers that got the best results at the ICCMA'21 competition, namely A-Folio-DPDB [2] for the SE-ST track and µ-Toksia [29] for the SE-PR track. As A-Folio-DPDB delegates the SE-ST problems to the µ-Toksia solver submitted at ICCMA'19, we finally used µ-Toksia (2019) for SE-ST problems. We chose to build communities of Erdös-Rényi graphs, since those graphs were already used to generate AFs and can be naturally generated as directed graphs. Communities were linked following a tree template (like ICCMA'21 instances). The linker processes in a way inspired by the $\mathcal{ER}$ generator : each possible edge from the source graph to the target graph is added with probability 0.2.

In the first part of our experiments, we sought which sizes of communities are small enough to be part of our graphs. We used crusti_g2io to generate single Erdös-Rényi graphs (by asking for an outer graph composed of a single node) with different number of nodes (from 100 to 1000) and probability for each edge to appear (0.1, 0.2 and 0.5). For each setting, we generated 10 different graphs by feeding the app with random seeds from 0 to 9 ; the computation times are averages of these 10 values, and a timeout of at least one makes the average be also timeout. We run experiments on machines equipped with Intel Xeon E5-2637 v4 processors and 128GB of RAM, and the timeout was fixed to 600s, as in ICCMA'21. Table 2 shows some experimental results.

First, we can note that for a given number of nodes, instances are more difficult for lower Erdös-Rényi probability values. This may be explained by the lower number of constraints, making preferred extensions admit more arguments, and stable extensions less common. This hypothesis would require further investigation, but is off-topic here

ER proba.	ER nodes	SE-ST (s)	SE-PR (s)
0,1	100	0,01	0,03
	200	3,13	9,14
	300	—	—
	400	—	—
0,2	100	0,02	0,02
	200	1,85	4,13
	300	13,87	22,91
	400	—	—
0,5	100	0,01	0,02
	200	0,10	0,07
	300	0,14	0,37
	400	0,23	4,11
	500	1,81	13,97
	600	4,28	16,56
	700	3,34	41,23
	800	6,72	74,41
	900	11,27	141,24
	1000	14,32	67,37

TABLE 2 – CPU time required by µ-Toksia 2019 (resp. 2021) to compute a single stable (resp. preferred) extension for different sizes of Erdös-Rényi graphs. CPU times are average of 10 values. If a timeout was reached for at least one graph, — is reported.

since we are only interested in the difficulty of the instances.

Communities of 100 arguments seem easy for both SE-ST and SE-PR, whatever the probability setting. With a setting of 0.1, the problems begin to require multiple seconds to be solved for 200 nodes ; this value should not be exceeded for instances involving several communities. A single community of 300 nodes cannot be solved in this context. With a setting of 0.2, the limit in terms of number of nodes to consider for multiple communities seems to be between 200 and 300 ; for this value, a single community requires more than 10 seconds for SE-ST, and more than 20s for SE-PR. A setting of 0.5 allows to generate instances with a single community of at least 1000 nodes. Interestingly we remarked that in this case, all instances admit stable extensions, which is not the case for the other probability settings. This indicates that these instances have a special structure that might make solvers work differently on them. Finally, as expected, the SE-PR problem takes more time to be solved than SE-ST.

Now that we have bounds on the size of the communities to consider, we can experiment the difficulty induced by the number of communities. We generated complete binary trees of Erdös-Rényi communities, where each community is linked to the ones associated with its children.

For this second experiment session, we considered Erdös-Rényi with nodes between 100 and 500 with the same three

2. `https://github.com/gorczyca/dp_on_dbs/tree/competition`

probability settings. We assumed the multiplicity of the communities would make the instances very hard for the 0.5 probability for more than 500 nodes per community. We considered outer tree heights from 3 to 9, making the outer graphs contain from 7 to 511 nodes. For each setting, 10 instances were generated with random seeds going from 0 to 9. We used to same machines and timeout than before. Figures 5 and 6 report the interesting parts of these new results. The plots on Figure 5 correspond to the results for the SE-ST track, while Figure 6 reports the results for SE-PR. For each figure, the three subfigures are each associated with a density setting (0.1, 0.2 and 0.5). For each subfigure, the average computation time is given on the y-axis, while the x-axis gives the number of communities ; the lines gives the different community sizes.

We first focus on the SE-ST results, given by the plots at Figures 5a, 5b and 5c. Concerning the results of µ-Toksia 2021 for the 0.1 probability setting (Figure 5a), we can observe that the problems are too easy when the number of nodes per community is lower than 200 (all solved in few seconds even for 511 communities) and too hard when it is above this value (such problems cannot be solved when there are more than 31 communities). Thus, this setting does not allow us to draw a clear frontier between the hard and the too-hard instances. This is also the case for the 0.5 probability setting (Figure 5c) for which the instances are surprisingly very difficult even for low values of community sizes and community counts. This is not an unexpected result since as noted below, these instances have a special structure that might prevent µ-Toksia to solve them. By the way, we discovered that µ-Toksia was not able to prove the absence of stable extension in any community-based instance with this density. If such instances are included in our benchmarks, then µ-Toksia may suffer from this special kind of instances. Fortunately, the 0.2 case (Figure 5b) perfectly fits our needs of frontier as it shows multiple settings of community sizes and counts are solvable but difficult (hundreds of seconds required to solve) namely the sets of 511 communities of size 225, the sets of 255 communities of size 250 and the sets of 63 communities of size 275.

Now, we discuss the SE-PR results, given by the plots at Figures 6a, 6b and 6c. Just like for SE-ST, the 0.1 probability setting (Figure 6a) does not seem to be an interesting value for us since little changes in community sizes makes the difficulty a lot higher : see e.g. the difference between communities of 175 nodes — almost difficult instances when there are 511 of them — and 200 nodes — where instances are too difficult for 255 communities. Things are a little better for the 0.2 probability (Figure 6b) when considering communities of size between 225 and 300, but the real interesting setting in this case if the 0.5 probability (Figure 6c). In this case, we can find at least three cases of different community sizes for which hard instances exist : the sets of 511 communities of 175 nodes, the sets of 255 communities of 300 nodes and the sets of 127 communities of 500 nodes.

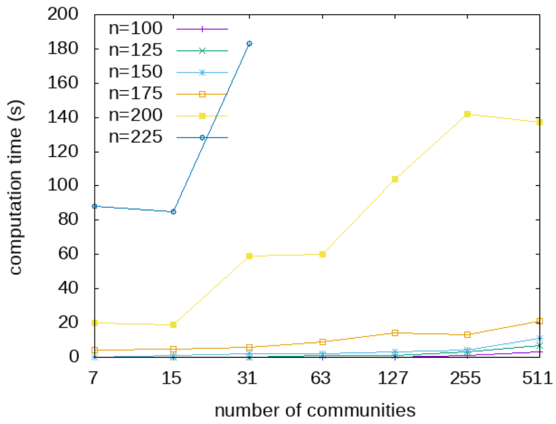
To conclude this section, it is worth noting that crusti_g2io generated the instances very fast. For the graph generation, we took advantage of machines with a higher number of processor cores. We dedicated to each process an Intel Xeon Gold 6248 (a 20-cores processor) and 192GB of RAM. The biggest instances we considered are the ones with 511 communities of 500 nodes with a probability setting of 0.5, for which the graph admits 255500 nodes and more than 89 millions edges. For these instances, the graph generation itself took less than 4s each. A little longer was necessary to translate the graphs into argumentation frameworks and store them using the (verbose) APX format on the hard disk. With these additional translation and writing times, the average wall-clock time was 19.62s.

8 Conclusion

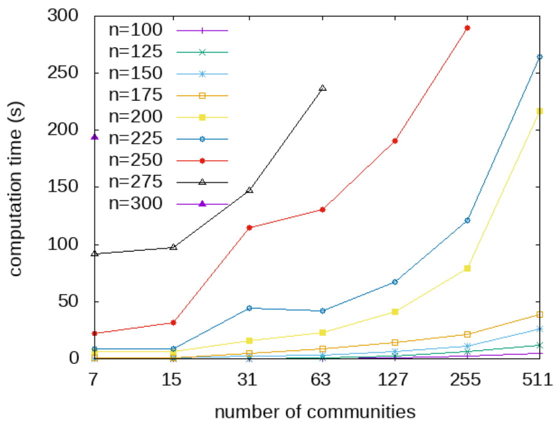
In this paper, we have defined a new approach for generating (directed or non-directed) graphs based on the concept of communities, which are graphs where some subparts of the graph are highly connected, but are loosely related to other subparts. Our approach uses a so-called inner/outer template, i.e. we first generate an outer graph representing the global structure of the graph, then in each node of the outer graph we generate an inner graph, and finally we use a linker to add edges between nodes of inner graphs which are connected in the outer graph structure. The proposed model is particularly generic and modular, since all the components (outer graph generator, inner graph generator and linker) can be replaced by other generators or linkers. Our model is particularly well suited for abstract argumentation, since large debates (i.e. large argumentation frameworks) can naturally be split into sub-debates which are only connected by a few arguments and attacks. We have described our open-source tool for the generation of graphs, and especially we have shown that this tool allows to generate meaningful argumentation framework instances with a level of difficulty for standard computational problems which can be adapted thanks to the choice of some parameters.

Several avenues for future work can be highlighted. Regarding the tool, a natural development direction is to design an even more generic framework, allowing several levels of nested graphs (i.e. the inner graph generator could generate graphs which also follow the inner/outer template). We also plan to improve the usability of the tool by describing the generation task in files (using e.g. the YAML or JSON format) instead of the command-line interface.

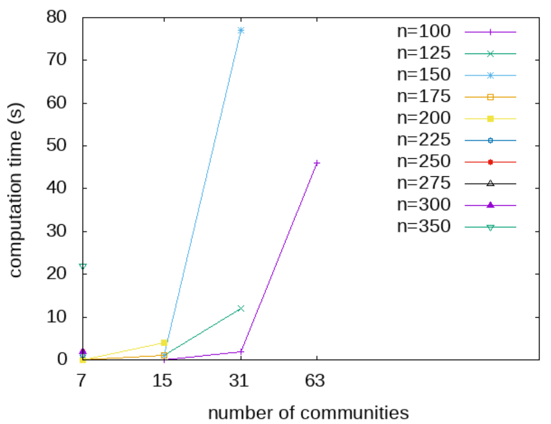
Regarding the issue of AF generation, we can improve the relevance of the tool by incorporating linkers which make sense in the context of abstract argumentation frameworks (for instance, we could add edges concerning in priority arguments which are skeptically accepted w.r.t. some given



(a) SE-ST, $\mathcal{ER}$ probability of 0.1

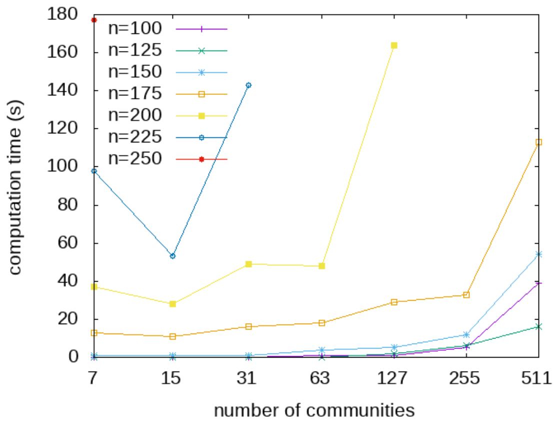


(b) SE-ST, $\mathcal{ER}$ probability of 0.2

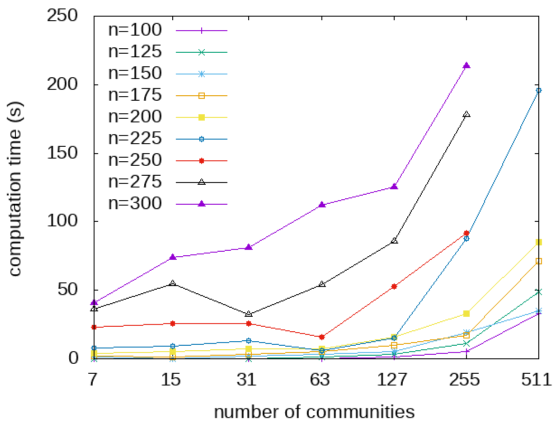


(c) SE-ST, $\mathcal{ER}$ probability of 0.5

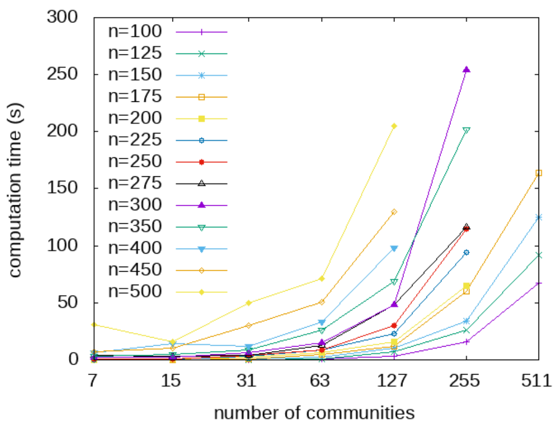
FIGURE 5 – CPU time (in seconds) required by μ-Toksia 2019 to compute a single stable extension for community graphs of different community sizes and different community count. CPU times are an average of 10 values.



(a) SE-PR, $\mathcal{ER}$ probability of 0.1



(b) SE-PR, $\mathcal{ER}$ probability of 0.2



(c) SE-PR, $\mathcal{ER}$ probability of 0.5

FIGURE 6 – CPU time (in seconds) required by μ-Toksia 2021 to compute a single preferred extension for community graphs of different community sizes and different community count. CPU times are an average of 10 values.

semantics). Another interesting future work consists in proposing generation models for more complex argumentation frameworks, which would require e.g. graphs with different kinds of edges or arguments (to incorporate supports [6] or incompleteness [27]) or graphs with weights associated with edges [13] or arguments [30].

Acknowledgements

This work has been partly supported by the CPER DATA Commode project from the "Hauts-de-France" Region, the ANR projects PING/ACK (ANR-18-CE40-0011) and AGGREEY (ANR-22-CE23-0005).

Références

[1] Barabási, A. et R. Albert: *Emergence of scaling in random networks*. Science, 286 :509–512, 1999.

[2] Baroni, P., M. Caminada et M. Giacomin: *Abstract Argumentation Frameworks and Their Semantics*. Dans *Handbook of Formal Argumentation*, pages 159–236. College Publications, 2018.

[3] Besnard, P. et A. Hunter: *Elements of Argumentation*. MIT Press, 2008, ISBN 978-0-262-02643-7.

[4] Beynier, A., Y. Chevaleyre, L. Gourvès, A. Harutyunyan, J. Lesca, N. Maudet et A. Wilczynski: *Local envy-freeness in house allocation problems*. Auton. Agents Multi Agent Syst., 33(5) :591–627, 2019.

[5] Bistarelli, S., F. Rossi et F. Santini: *ConArg : A Tool for Classical and Weighted Argumentation*. Dans *Proc. of COMMA 2016*. IOS Press, 2016.

[6] Cayrol, C. et M. C. Lagasquie-Schiex: *Bipolarity in argumentation graphs : Towards a better understanding*. Int. J. Approx. Reason., 54(7) :876–899, 2013.

[7] Cerutti, F., M. Giacomin et M. Vallati: *Generating Challenging Benchmark AFs*. Dans *Proc. of COMMA 2014*, 2014.

[8] Cerutti, F., M. Giacomin et M. Vallati: *Generating Structured Argumentation Frameworks : AFBenchGen2*. Dans *Proc. of COMMA 2016*, 2016.

[9] Chung, F. et Li. Lu: *The Average Distance in a Random Graph with Given Expected Degrees*. Internet Math., 1(1) :91–113, 2003.

[10] Cochez, M., M. Croitoru, P. Marquis et S. Rudolph (rédacteurs): *Proceedings of GKR 2020*, tome 12640 de *Lecture Notes in Computer Science*, 2021.

[11] Cooper, M. C., A. Herzig, F. Maffre, F. Maris et P. Régnier: *The epistemic gossip problem*. Discret. Math., 342(3) :654–663, 2019.

[12] Dung, P. M.: *On the Acceptability of Arguments and its Fundamental Role in Nonmonotonic Reasoning, Logic Programming and n-Person Games*. Artif. Intell., 77(2) :321–358, 1995.

[13] Dunne, P. E., A. Hunter, P. McBurney, S. Parsons et M. Wooldridge: *Weighted argument systems : Basic definitions, algorithms, and complexity results*. Artif. Intell., 175(2) :457–486, 2011.

[14] Dvorák, W. et P. E. Dunne: *Computational Problems in Formal Argumentation and their Complexity*. Dans *Handbook of Formal Argumentation*, pages 631–688. College Publications, 2018.

[15] Dvorák, W., S. A. Gaggl, A. Rapberger, J. P. Wallner et S. Woltran: *The ASPARTIX System Suite*. Dans *Proc. of COMMA 2020*, 2020.

[16] Dvorák, W., M. Järvisalo, J. P. Wallner et S. Woltran: *Complexity-sensitive decision procedures for abstract argumentation*. Artif. Intell., 206 :53–78, 2014.

[17] Edunov, S., D. Logothetis, C. Wang, A. Ching et M. Kabiljo: *Generating Synthetic Social Graphs with Darwini*. Dans *Proc. of ICDCS*, pages 567–577, 2018.

[18] Erdös, P. et A. Rényi: *On random graphs. I.* Publicationes Mathematicae, 6 :290–297, 1959.

[19] Geilen, N. et M. Thimm: *Heureka : A General Heuristic Backtracking Solver for Abstract Argumentation*. Dans *Proc. of TAFA 2017*, pages 143–149, 2017.

[20] Girvan, M. et M. Newman: *Community structure in social and biological networks*. Proc. of the NAS of the USA, 99(12) :7821–7826, 2002.

[21] Heinrich, M.: *The MatrixX Solver For Argumentation Frameworks*. CoRR, abs/2109.14732, 2021.

[22] Kinder, L., M. Thimm et B. Verheij: *A Labeling Based Backtracking Solver for Abstract Argumentation*. Dans *Proc. of SAFA 2022*, pages 111–123, 2022.

[23] Kolda, T., A. Pinar, T. Plantenga et C. Seshadhri: *A Scalable Generative Graph Model with Community Structure*. SIAM J. Sci. Comput., 36(5), 2014.

[24] Lagniez, J. M., E. Lonca et J. G. Mailly: *CoQuiAAS : A Constraint-Based Quick Abstract Argumentation Solver*. Dans *Proc. of ICTAI 2015*, pages 928–935, 2015.

[25] Lagniez, J. M., E. Lonca, J. G. Mailly et J. Rossit: *Design and Results of ICCMA 2021*. CoRR, abs/2109.08884, 2021.

[26] Leskovec, J., K. Lang, A. Dasgupta et M. Mahoney: *Statistical properties of community structure in large social and information networks*. Dans *Proc. of WWW 2008*, pages 695–704, 2008.

[27] Mailly, J. G.: *Yes, no, maybe, I don't know : Complexity and application of abstract argumentation with incomplete knowledge*. Argument Comput., 13(3) :291–324, 2022.

[28] Motik, B., B. Cuenca Grau, I. Horrocks et U. Sattler: *Representing ontologies using description logics, description graphs, and rules*. Artif. Intell., 173(14) :1275–1309, 2009.

[29] Niskanen, A. et M. Järvisalo: *μ-toksia : An Efficient Abstract Argumentation Reasoner*. Dans *Proc. of KR 2020*, pages 800–804, 2020.

[30] Rossit, J., J. G. Mailly, Y. Dimopoulos et P. Moraitis: *United we stand : Accruals in strength-based argumentation*. Argument Comput., 12(1) :87–113, 2021.

[31] Watts, D. et S. Strogatz: *Collective dynamics of "small-world" networks*. Nature, 393 :440–442, 1998.

[32] Wu, X., X. Ying, K. Liu et L. Chen: *A Survey of Privacy-Preservation of Graphs and Social Networks*. Dans *Managing and Mining Graph Data*, pages 421–453. 2010.