Comparing Unsupervised Algorithms to Construct Argument Graphs

Mirko Lenz[1,*], Lorik Dumani[1] and Premtim Sahitaj[1]

[1]Trier University, Universitätsring 15, 54296 Trier, Germany

Abstract

Computational argumentation has gained considerable attention in recent years. Various areas have been addressed, such as extracting arguments from natural language texts into a structured form in order to store them in an argument base, determining stances for arguments with respect to topics, determination of inferences from statements, and much more. After so much progress has been made in the isolated tasks, in this paper we address the next level and aim to advance the automatic generation of argument graphs. To this end, we investigate various unsupervised methods for constructing the graphs and measure the performance with different metrics on three different datasets. Our implementation is publicly available on GitHub under the permissive MIT license.

Keywords

argument mining, computational argumentation, clustering

1. Introduction

Computational argumentation is a research subfield of NLP which has received much attention and made great progress in recent years. Previous work investigated, among others, extracting Argumentative Discourse Units (ADUs) —that is, the smallest units of argumentation which may range from multiple words to entire paragraphs— in unstructured natural language texts [1], determining the stances of arguments into supporting or rejecting towards a standpoint, or retrieving a ranked list of the most convincing arguments to a query entered by the user [2].

While most prior work dealt with such isolated tasks, only a few works so far tackled the merging of multiple tasks to generate complete argument graphs that represent the structure of a text as well as their views in a clear, concise, and compact manner [3, 4]. The need for argument graphs is justified not only by argument search engines [5, 6] that obtain their arguments from such structures, but also by the many publicly available datasets such as the argumentative essays dataset [2] as well as the argument graphs available on AIFdb. We argue that after so much progress in this area, it is now time to combine these efforts and shift the focus to the automatic generation of argument graphs.

Proceedings of the Workshop on Text Mining and Generation (TMG) co-located with the 45rd German Conference on Artificial Intelligence (KI 2022), September 19, 2022, Trier (Virtual), Germany

*Corresponding author.

info@mirko-lenz.de (M. Lenz); dumani@uni-trier.de (L. Dumani); sahitaj@uni-trier.de (P. Sahitaj)

https://www.mirko-lenz.de (M. Lenz)

0000-0002-7720-0436 (M. Lenz); 0000-0001-9567-1699 (L. Dumani); 0000-0003-3908-5681 (P. Sahitaj)

CEUR Workshop Proceedings (CEUR-WS.org)

In this paper, we explore several unsupervised methods to create argument graphs and evaluate these on three datasets which are completely different in structure as well as in size and domain. Since our code is publicly available on GitHub under the permissive MIT license, interested researchers can build upon our work and use these methods as a baseline.[1] Each of them expects a set of ADUs as input to produce a fully connected tree structure where each node is an ADU. As such, we can ignore the original text of an argument—which may not always be available anyway—and leave the detection of ADUs to future work. We assume that each ADU supports/attacks exactly one other ADU and that there exists one ADU acting as the root of an argument graph—the so-called major claim.

Our contributions are the following: (i) Seven algorithms for constructing argument graphs in pseudocode together with a reference implementation in Python. (ii) A set of metrics for evaluating our argument graphs each focused on different aspects. (iii) An experimental evaluation on three diverse datasets that may be used as a baseline for future work.

Next, we discuss related work in section 2. In particular, we address similarities and differences to our work. In section 3 we describe the seven methods we use utilizing pseudo code and in section 4 we discuss the evaluation setup and results. In section 5 we conclude the paper and provide starting points for future work.

2. Foundations and Related Work

The general approach for argument graph construction in the literature is based on the utilization of information extracted in the previous tasks. Following the concept of claim-premise model [7, 8], the approach is to classify potentially argumentative text spans as either claim, premise, major claim, or non-argumentative [3, 9, 10]. The *claim* depicts a potentially controversial viewpoint. The arguer tries to support it with *premises*, which serve as evidence. Frequently, there is also a core viewpoint in texts, the *major claim*. Each of these three terms is a certain kind of an ADU—the smallest unit of argumentation introduced earlier.

Stab and Gurevych [10] describe an argument graph annotation process specifically for the Persuasive Essays dataset because of their structured constituency. Arguments are defined as a single claim with several premises (one-claim approach). A paragraph can consist of multiple arguments. Premises are allowed to be connected only to a claim of the same paragraph, but not to claims outside of this defined space. All claims are linked to the major claim. An exemplary graph from this dataset is depicted in fig. 1. It will also be part of our experimental evaluation in section 4.

Similar approaches are implemented by Persing and Ng [3, 9] while additional rules are exploited and utilized for the graph construction process. As part of a larger argument mining pipeline, Lenz et al. [4] describe and evaluate three graph construction approaches. Due to the larger scope of their work, they were able to utilize additional information during graph construction—for instance, the probability that one ADU is supported/attacked by another one. An interesting approach to the graph construction task has been published by Gemechu and Reed [11] where the authors identify different functional components (i.e., concepts, aspects,

[1] https://github.com/recap-utr/argmining-clustering

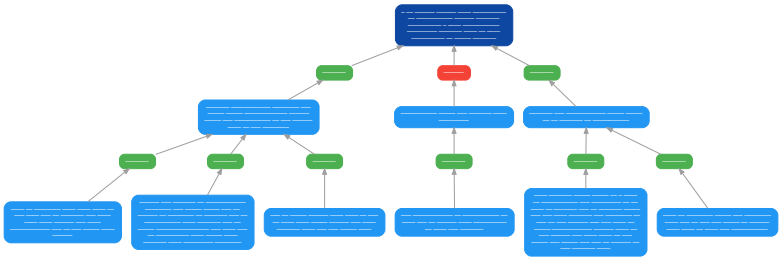


Figure 1: First argument graph taken from the corpus of persuasive essays [10]. The textual content of the nodes has been removed since we are interested in the structure.

opinions) from argumentative statements and utilize these features to connect argument structures into the final argument graph. We refer the interested reader to the surveys conducted by Lawrence and Reed [1] as well as Schaefer and Stede [12] for more details. The methods that we introduce in this paper follow the work of Lenz et al. [4] and can be differentiated from other research by the fact that we aim to avoid domain-specific construction rules, e.g. restricting the connection of arguments by their (predicted) classification type.

3. Algorithms

In this section we will introduce our algorithms for constructing argument graphs from a given set of ADUs. In addition to the aforementioned implementations in Python (used for our evaluation in section 4), we also provide pseudocode for each of them to aid understanding implementing them in other languages.

Before diving in, we will first define some common symbols used in the pseudocode. All algorithms are passed a set of ADUs $A = \{a_1, \ldots, a_n\}$ where each $a_i \in A$ is string. We denote the ADU being major claim of an argument graph as $a_{\text{mc}} \in A$. Each algorithm returns this major claim a_{mc} together with a list of relations $R = \{r_1, \ldots, r_n\}$. Each relation $r_i = (a_{\text{prem}}, a_{\text{claim}}) \in A \times A$ is a tuple containing a premise a_{prem} that is connected to a claim a_{claim} via an inference. Some of our proposed algorithms make use of clustering methods where $C = \{c_1, \ldots, c_n\}$ denotes the set of clusters C that forms a partition of A. Each cluster $c_i \in C$ is defined as a set of ADUs (i.e., $c_i \subseteq A$).

We also use some functions that are crucial for most of the algorithms. Each ADU $a \in A$ has a feature vector that can be accessed using the function $\text{vec}(a)$ and is defined via a embedding model. This vector space makes it possible to compute a centroid m for each cluster c_i at its center point. These vectors are also used to compute the cosine similarity between ADUs—as an abbreviation, we use the function $\text{sim}(a_i, a_j) = \cos(\text{vec}(a_i), \text{vec}(a_j))$.

Our algorithms are able to predict the central major claim a_{mc} without any further information. In addition, it is also possible to set a_{mc} manually. For details on that matter we refer the interested reader to our reference implementation mentioned above.

Agglomerative With algorithm 3.1, we present a graph construction strategy that is based on agglomerative clustering and retains its hierarchical structures throughout the construction

Algorithm 3.1 Agglomerative-clustering-based algorithm for constructing argument graphs.

procedure AGGLOMERATIVE(ADUs A)
  $C \leftarrow \{\{a\} \mid a \in A\}$
  $R \leftarrow \emptyset$
  while $|C| > 1$ **do**
    $i, j \leftarrow$ find the two most similar clusters in C ▷ Ordered s.t. $|c_i| < |c_j|$
    $a_{\text{prem}} \leftarrow$ root of smaller cluster $A[i]$
    $a_{\text{merge}} \leftarrow A[j]$ root ADU of larger cluster
    $a_{\text{claim}} \leftarrow$ ADU of larger cluster $a \in c_j$ having the highest similarity to premise a_{prem}
    delete entries i, j of sets C and A
    $R \leftarrow R \cup \{(a_{\text{prem}}, a_{\text{claim}})\}$
    $C \leftarrow C \cup$ MERGE(Clusters c_i, c_j)
    $A \leftarrow A \cup \{a_{\text{merge}}\}$
  $a_{\text{mc}} \leftarrow$ root node of relations R
  return R, a_{mc}

process. The graph construction strategy utilizes a selection metric based on cosine similarity between embedding representations of ADUs to mimic an hypothetical concept of relatedness. The hypothesis follows the idea that more similar ADUs have a higher chance to target the same topics and thus provide related argumentative information. Each of the n argumentative components starts within its own cluster (algorithm 3.1). Clusters are iteratively merged on the basis of a given criterion—in our case, average linkage. From the pair of most similar clusters, we identify the smaller and larger cluster. For the purpose of constructing more dense argument graphs structures, we consider the size of the clusters to be merged. The smaller cluster proposes the root of its respective sub-graph as premise (algorithm 3.1). Then, from the larger cluster we identify the ADU that maximizes the similarity to the previously identified premise and denote it as claim. Finally, we draw a relation from premise to claim (algorithm 3.1) which connects the sub-graphs of the two respective clusters. We update our lists and continue the iteration. Merging is repeated a total of $n - 1$ times and per step only two clusters are merged until all ADUs are in the same cluster. Finally, we are left with a single connected argument graph.

DENSITY Algorithm 3.2 is a cluster-based approach that uses HDBSCAN [13, 14] internally. HDBSCAN is optimized for scenarios with a high density of data points as well as noisy data—both of which are relevant for embeddings of natural language texts. First, the underlying algorithm is run (algorithm 3.2) to construct the hierarchical/tree-based cluster structure. The following steps are different for *natural* clusters—that is, a cluster containing actual ADUs—and *synthetic* clusters—that is, a cluster with a single element that is not part of the set of ADUs A and thus only created to allow nesting of other clusters. In the former case (algorithm 3.2), we define the claim to be the ADU having the highest similarity to all others and create relations between it and the remaining premises. In the latter case, we first need to resolve connections between two synthetic clusters that have no natural cluster as a direct child. More specifically, this means

Algorithm 3.2 Density-based algorithm for constructing argument graphs.

procedure DENSITY(ADUs A)
  $R \leftarrow \emptyset$
  $a_{\text{mc}} \leftarrow$ NULL
  $C_{\text{tree}} \leftarrow$ HDBSCAN($A, n_{\text{clusters}} \leftarrow 2, n_{\text{samples}} \leftarrow 1$)
  for all natural clusters $c_{\text{nat}} \in C_{\text{tree}}$ **do** ▷ First deal with clusters containing ADUs
    $a_{\text{claim}} \leftarrow a \in c_{\text{nat}}$ s.t. $\text{sim}(a, a') > \text{sim}(a', a'') \forall a'' \in c_{\text{nat}}$
    if c_{nat} is the first cluster in C **then**
      $a_{\text{mc}} \leftarrow a_{\text{claim}}$
    $R \leftarrow R \cup \{(a_{\text{prem}}, a_{\text{claim}})\}$ for all $a_{\text{prem}} \in A'$
  contract connections between two synthetic clusters having no natural clusters
  for all synthetic clusters $c_{\text{synt}} \in C_{\text{tree}}$ **do** ▷ Now, we can connect the nested clusters
    $R \leftarrow R \cup \{(\text{claim of} c_{\text{nat}}, \text{claim of} c_{\text{synt}})\}$
  return R, a_{mc}

that we remove two relations from the hierarchical cluster structure C_{tree} and replace it with a single relation (algorithm 3.2). We then can continue by connecting the claims of the synthetic clusters to the claims of the natural clusters (algorithm 3.2).

DIVIDE The main idea behind algorithm 3.3 is to divide it into smaller subproblems, solve them individually, and aggregate these solutions later on—the so-called *divide and conquer* approach [15]. More specifically, we use a relatively straightforward clustering algorithm—k-Means [16]—to divide the set of ADUs into smaller chunks (algorithm 3.3). This procedure is recursively executed until there are at most n_{leaf} ADUs in a cluster (algorithm 3.3). This hyperparameter may be set to an arbitrary value, but in our experiments, we set it to 3 since most argumentation schemes [17] are composed of one claim and two premises. Within each cluster, we select the ADU being most similar to all remaining ones to be the claim, the others are used as premises for that claim. The k-Means algorithm expects a fixed number k corresponding to the number of clusters that shall be created. We run the algorithm using a range of different k's and utilize the well-known silhouette score [18] to determine the optimal number of clusters. In order to limit the computational overhead of this approach in some extent, we restrict the maximum value for k to half the number of available ADUs. At the end, we receive one argument graph where each branch corresponds to a recursive function call.

FLAT Algorithm 3.4 is mainly implemented as a baseline/comparison algorithm and is borrowed from Lenz et al. [4]. First, the major claim is identified using the same heuristic as for the algorithm SIM (algorithm 3.4). Then, all remaining ADUs are directly connected to the major claim, resulting in a graph with only two levels (algorithm 3.4).

ORDER The idea of algorithm 3.5 is that we sort ADUs based on their similarity to the major claim and process them in that order with some degree of freedom. This uses the assumption that ADUs being similar to the major claim should be connected to it as claims. More precisely,

Algorithm 3.3 Divide-based algorithm for constructing argument graphs.

procedure DIVIDE(ADUs A, Centroid $m \leftarrow$ NULL, Maximum number of leaf nodes $n_{\text{leaf}} \leftarrow 3$)
  $R \leftarrow \emptyset$
  if $m =$ NULL **then**
    $m \leftarrow$ arithmetic mean of $\text{vec}(a)$ for all $a \in A$
  $a_{\text{claim}} \leftarrow a \in A$ having the highest similarity to all other ADUs
  $A \leftarrow A \setminus \{a_{\text{claim}}\}$
  if $|A| \leq n_{\text{leaf}}$ **then** ▷ Termination criterion for recursive calls
    return $A \times \{a_{\text{claim}}\}, a_{\text{claim}}$ ▷ Connect all remaining ADUs to the claim a_{claim}
  $k_{\min} \leftarrow 2$ ▷ We need at least two clusters ...
  $k_{\max} \leftarrow \max(|A| \div 2, k_{\min}) + 1$ ▷ ...and at most half the number of ADUs
  $k_{\text{opt}} \leftarrow$ value of $k \in [k_{\min}, k_{\max}]$ resulting in the highest silhouette score
  for all centroids $m' \in$ KMEANS(A, k_{opt}) **do**
    $A' \leftarrow$ subset of A where each $a \in A$ is assigned to centroid m'
    $R', a'_{\text{claim}} \leftarrow$ DIVIDE(A', m') ▷ Call function on subset of ADUs
    $R \leftarrow R \cup R' \cup \{(a'_{\text{claim}}, a_{\text{claim}})\}$
  return R, a_{claim}

Algorithm 3.4 Flat algorithm for constructing argument graphs.

procedure FLAT(ADUs A)
  $R \leftarrow \emptyset$
  $a_{\text{mc}} \leftarrow a \in A$ having the highest similarity to all other ADUs
  for all $a \in A \setminus \{a_{\text{mc}}\}$ **do**
    $R \leftarrow R \cup \{(a, a_{\text{mc}})\}$
  return R, a_{mc}

we select the candidate in the k-neighborhood (algorithm 3.5) as premise that maximizes the similarity between claim and premise for each queued claim. The selected premise is then removed from the queue and connected to the claim. Finally, we update the variables and continue the iteration until all ADUs are connected to one graph structure. The approach is applied TOP-DOWN and is viable from any arbitrary major claim position.

RANDOM The purpose of algorithm 3.6 is to serve as an evaluation baseline. We do not consider similarities at all, but follow a completely random approach. Consequently, the major claim is chosen randomly (algorithm 3.6). For all remaining ADUS, we randomly select one that is not yet assigned together with one that is part of the graph (algorithm 3.6).

SIM Algorithm 3.7 may be viewed as a more simplistic version of ORDER. In essence, we leave out the sorting step of ADUs to the major claim. We start by selecting the ADU having the highest similarity to all others as the major claim (algorithm 3.7). Then, we iterate over the

Algorithm 3.5 Ordering-based algorithm for constructing argument graphs.

procedure ORDER(ADUs A)
  $R \leftarrow \emptyset$
  $a_{\text{mc}} \leftarrow a \in A$ having the highest similarity to all other ADUs
  $A \leftarrow A \setminus \{a_{\text{mc}}\}$
  SORT(A) based on similarity of an ADU $a \in A$ to the major claim a_{mc}
  $A' \leftarrow \{a_{\text{mc}}\}$
  while $|A| > 0$ **do**
    $a_{\text{prem}}, a_{\text{claim}} \leftarrow$ pair of ADUs in $A \times A'$ having the highest similarity
      ▷ s.t. CONSTRAINT($a_{\text{prem}}, a_{\text{claim}}$) holds
    $R \leftarrow R \cup \{(a_{\text{prem}}, a_{\text{claim}})\}$
    $A \leftarrow A \setminus \{a_{\text{prem}}\}$
    $A' \leftarrow A' \cup \{a_{\text{prem}}\}$
  return R, a_{mc}
procedure CONSTRAINT(ADUs $a_{\text{prem}}, a_{\text{claim}}$, Threshold $k \leftarrow 2$)
  return $|\text{pos}(a_{\text{claim}}) - \text{pos}(a_{\text{prem}})| \leq k$ ▷ pos denotes the position of the ADUs in algorithm 3.5

Algorithm 3.6 Random algorithm for constructing argument graphs.

procedure RANDOM(ADUs A)
  $R \leftarrow \emptyset$
  $a_{\text{mc}} \leftarrow \text{random}(A)$
  $A \leftarrow A \setminus \{a_{\text{mc}}\}, \quad A' \leftarrow \{a_{\text{mc}}\}$
  while $|A| > 0$ **do**
    $a_{\text{prem}}, a_{\text{claim}} \leftarrow \text{random}(A \times A')$
    $R \leftarrow R \cup \{(a_{\text{prem}}, a_{\text{claim}})\}$
    $A \leftarrow A \setminus \{a_{\text{prem}}\}, \quad A' \leftarrow A' \cup \{a_{\text{prem}}\}$
  return R, a_{mc}

remaining ADUs (algorithm 3.7) and compute the similarity between all ADUs that are not part of the graph to the ones already added by constructing the cross product of the sets A and A' (algorithm 3.7). A new relation is then created between the two ADUs selected in the previous step (algorithm 3.7) until all ADUs are connected.

4. Evaluation

Having presented our proposed algorithms for constructing argument graphs, we will now conduct an experimental evaluation with the goal of discussing the advantages as well as the drawbacks of them. We will briefly describe our experimental setup, present the results, and discuss them afterwards.

Algorithm 3.7 Similarity-based algorithm for constructing argument graphs.

procedure SIM(ADUs A)
  $R \leftarrow \emptyset$
  $a_{\text{mc}} \leftarrow a \in A$ having the highest similarity to all other ADUs
  $A \leftarrow A \setminus \{a_{\text{mc}}\}$
  $A' \leftarrow \{a_{\text{mc}}\}$
  while $|A| > 0$ **do**
    $a_{\text{prem}}, a_{\text{claim}} \leftarrow$ pair of ADUs in $A \times A'$ having the highest similarity
    $R \leftarrow R \cup \{(a_{\text{prem}}, a_{\text{claim}})\}$
    $A \leftarrow A \setminus \{a_{\text{prem}}\}$
    $A' \leftarrow A' \cup \{a_{\text{prem}}\}$
  return R, a_{mc}

4.1. Datasets

In order to obtain meaningful results, we evaluated our algorithms on a total of three datasets. The first is the *Microtexts* dataset from Peldszus and Stede [19] as offered by the AIFdb project.[2] This comprises a total of 110 argument graphs, each containing 4 ADUs on average. The second dataset by Stab and Gurevych [10] includes a total of 402 *persuasive essays*. Herein, the structure is more complex than for the Microtexts because, as already described in section 2, they consist of three levels, namely the major claim level, the claim level, and the premise level. The average number of ADUs per graph here is 14. The last dataset is a subset of the *Kialo* dataset containing all graphs obtained by Lenz et al. [4] that do not exceed a maximum size of 100 KB and is once again more complex, comprising a total of 90 argument graphs with an average of 41 ADUs per graph.

4.2. Metrics

As measures for graphs yield different scores depending on the goal, we use multiple:

1. The duration t_{ms} (measued in ms) needed to reconstruct the graph.

2. The graph edit similarity sim_{edit} that computes the number edit of operations needed to transform one graph to another.[3] We transform this $\text{dist}_{\text{edit}}$ to a similarity score in $[0, 1]$ by computing

$$\text{sim}_{\text{edit}}(g_1, g_2) = 1 - \frac{\text{dist}_{\text{edit}}(g_1, g_2)}{\text{dist}_{\text{max}}(g_1, g_2)} \tag{1}$$

   with dist_{max} denoting the maximum number of edit operations—that is, removing every element of one graph and adding all elements of the other. In our paper, the nodes are identical, meaning that edit operations only operate on edges.

[2] http://corpora.aifdb.org/Microtext
[3] We utilize the library `graphkit-learn`: https://github.com/jajupmochi/graphkit-learn

3. The Jaccard similarity sim_J assesses the number of correctly predicted edges as

$$\mathrm{sim}_J(g_1, g_2) = J\big(\mathrm{edges}(g_1), \mathrm{edges}(g_2)\big) \tag{2}$$

with $\mathrm{edges}(g)$ being the set of edges of argument graph g and J being the Jaccard index that for the two sets A, B is defined as [20]

$$J(A, B) = \frac{|A \cap B|}{|A \cup B|}. \tag{3}$$

4. The major claim agreement $\mathrm{sim}_{\mathrm{mc}}$ is a binary metric that is defined as

$$\mathrm{sim}_{\mathrm{mc}}(g_1, g_2) = \begin{cases} 1, & \text{if } \mathrm{mc}(g_1) = \mathrm{mc}(g_2), \\ 0, & \text{otherwise.} \end{cases} \tag{4}$$

with $\mathrm{mc}(g)$ being the major claim of argument graph g.

5. The metric $\mathrm{sim}_{\mathrm{depth}}$ is based on the average tree depth [21] that is a *visual* indicator whether the vertical structure of two graphs overlaps. It is defined as

$$\mathrm{sim}_{\mathrm{depth}}(g_1, g_2) = 1 - \frac{|\mathrm{depth}(g_1) - \mathrm{depth}(g_2)|}{\max(\mathrm{depth}(g_1), \mathrm{depth}(g_2))} \tag{5}$$

with $\mathrm{depth}(g)$ denoting the average tree depth of an argument graph g.

6. Similar to $\mathrm{sim}_{\mathrm{depth}}$, the metric $\mathrm{sim}_{\mathrm{breadth}}$ functions as a *visual* indicator of a graph's horizontal structure. We determine the mean average error of the number of nodes each graph has on each level and define the operators $\mathrm{level}_i(g)$ for retrieving the number of nodes on level i of argument graph g as well as $\mathrm{levels}(g)$ for determining the total levels of graph g. Using $n = \max(\mathrm{levels}(g_1), \mathrm{levels}(g_2))$ and the equation

$$\mathrm{dist}_{\mathrm{breadth}}(g_1, g_2) = \sum_{i=1}^{n} \frac{|\mathrm{level}_i(g_1) - \mathrm{level}_i(g_2)|}{n}, \tag{6}$$

we can now define $\mathrm{sim}_{\mathrm{depth}}$ as

$$\mathrm{sim}_{\mathrm{breadth}}(g_1, g_2) = 1 - \frac{\mathrm{dist}_{\mathrm{breadth}}(g_1, g_2)}{\sum_{i=1}^{\mathrm{levels}(g_1)} \mathrm{level}_i(g_1)} \tag{7}$$

4.3. Implementation Notes

We implemented the project with Python[4] and seeded random states with the integer 0 to get deterministic results. We also created a Docker container that makes it straightforward to reproduce our results. For the computation of the embeddings we use the popular library `spaCy`. We conduct experiments using the standard model `en_core_web_lg` based on plain word embeddings as well as the more advanced `en_core_web_trf` utilizing transformer-based embeddings.[5] To parse the argument graph corpora, we utilize the `arguebuf` library [22].

Table 1
Evaluation results of the proposed algorithms when applied to different datasets. The best scores for each metric in a dataset are marked in bold.

Dataset	Algorithm	t_{ms}	sim_{edit}	sim_{J}	sim_{mc}	$\text{sim}_{\text{breadth}}$	$\text{sim}_{\text{depth}}$
Microtexts	Agglomerative	0.932	0.755	0.144	**0.209**	0.807	0.776
	Density	1.576	0.795	**0.148**	**0.209**	0.861	0.872
	Divide	38.566	0.741	0.112	0.136	0.816	0.834
	Flat	**0.040**	**0.830**	0.120	0.145	**0.862**	**0.876**
	Order	0.070	0.745	0.110	0.145	0.826	0.793
	Random	0.043	0.728	0.081	0.045	0.732	0.654
	Sim	0.069	0.748	0.117	0.145	0.857	0.833
Essays	Agglomerative	26.261	0.545	0.074	**0.264**	0.859	0.757
	Density	2.543	0.596	0.057	0.097	0.824	0.830
	Divide	312.668	0.540	0.054	0.241	0.850	0.815
	Flat	**0.053**	**0.671**	0.061	0.236	0.623	0.648
	Order	0.165	0.549	**0.088**	0.236	0.864	0.613
	Random	0.077	0.554	0.041	0.000	0.840	0.750
	Sim	0.422	0.549	0.062	0.236	**0.877**	**0.843**
Kialo	Agglomerative	1735.590	0.448	**0.037**	0.011	0.887	0.615
	Density	5.584	0.500	0.022	**0.056**	0.882	**0.826**
	Divide	2335.256	0.439	0.022	0.000	0.855	0.699
	Flat	**0.096**	**0.619**	0.010	0.000	0.721	0.577
	Order	1.763	0.474	0.073	0.000	**0.924**	0.363
	Random	0.173	0.448	0.016	0.000	0.895	0.670
	Sim	8.043	0.440	0.028	0.000	0.906	0.752

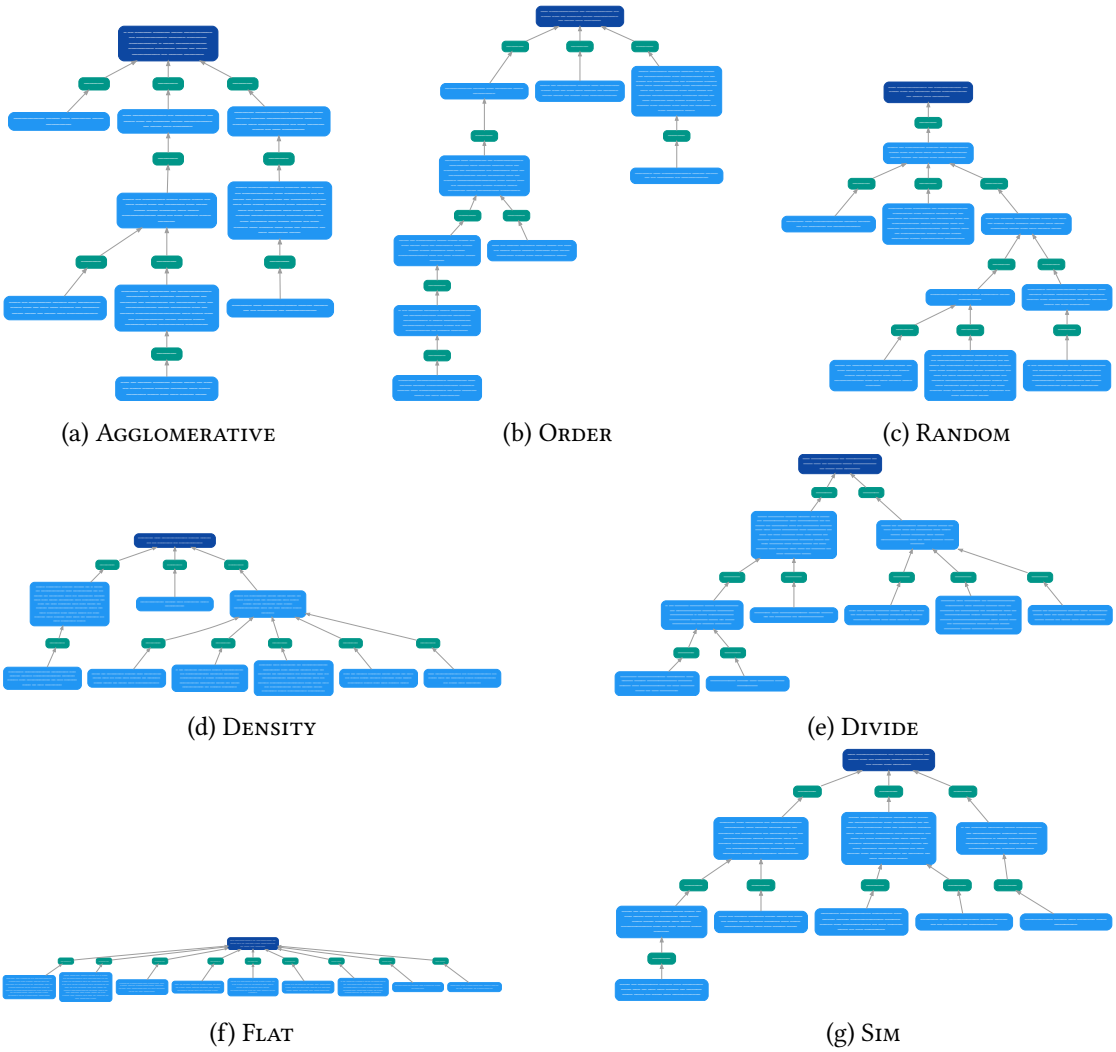
4.4. Results

Table 1 shows the performance of our methods on the three datasets with the different evaluation metrics. The main findings are that (i) the baselines Flat performs comparatively poorly for all three datasets, (ii) while the comparison method Random yields average results (as expected), our methods perform better across the board, (iii) more complex argument graphs are harder to generate automatically, and (iv) contrary to our expectation, Divide is computationally expensive (most likely caused by multiple runs to determine the best silhouette score). We conducted the same experiments with the transformer-based model `en_core_web_trf` and also when using the major claim from the gold standard, but did not observe major improvements w.r.t. the metrics.

4.5. Case Study

Since the creation of these rather complex graph structures is quite subjective [23], quantitative results as shown in table 1 only tell a part of the story. To complement our findings, fig. 2 shows

[4] https://github.com/recap-utr/argmining-clustering
[5] https://spacy.io/



(a) Agglomerative (b) Order (c) Random

(d) Density (e) Divide

(f) Flat (g) Sim

Figure 2: Case study of our algorithms when applied to the ADUs of fig. 1.

the result of all seven algorithms when applied to the set of ADUs available in the exemplary graph shown in fig. 1. The corresponding evaluation metrics are depicted in table 2. Please note that the text is not readable due to size constraint of this publication. Nonetheless, fig. 2 still provides valuable insights into the different graph structures that are created by our proposed algorithms. Compared to the original graph (fig. 1), we observe that (i) the structures generated by Flat, Agglomerative, Order, and Random greatly differ w.r.t. the number of levels, (ii) the only one having the same number of levels is Density despite having a different edge distribution, and (iii) only one of the approaches is able to detect the correct major claim. A final takeaway from this comparison is that the metrics used for our comparison give a rather complete overview of the results. Using sim_{edit}, we can tell that the number of correctly drawn edges is low for all approaches, while there are with high values for $\text{sim}_{\text{breadth}}$ and $\text{sim}_{\text{depth}}$

Table 2
Evaluation results for graphs depicted in fig. 2. The best scores for each metric are marked in bold.

Algorithm	t_{ms}	sim_{edit}	sim_{J}	sim_{mc}	$\text{sim}_{\text{breadth}}$	$\text{sim}_{\text{depth}}$
Agglomerative	5.798	0.579	**0.125**	**1.000**	0.822	0.800
Density	4.826	0.632	0.059	0.000	**1.000**	**0.952**
Divide	150.204	0.579	**0.125**	0.000	0.889	0.900
Flat	0.353	0.684	0.000	0.000	0.556	0.667
Order	0.430	0.421	0.059	0.000	0.852	0.800
Random	**0.255**	0.526	0.000	0.000	0.778	0.714
Sim	0.562	0.395	0.059	0.000	0.944	0.938

(e.g., Density) that still look rather similar to the original.

5. Conclusion and Future Work

In this paper, we have successfully implemented a series of unsupervised algorithms for constructing highly structured argument graphs from an unordered set of ADUs. In addition, we introduced metrics that assess different properties of the resulting graphs and provide a comprehensive numerical view on the results. Lastly, we have demonstrated that our approaches create vastly different graphs from the same input data. The construction of argument graphs still is a rather subjective topic where even humans may not share the same opinion. Thus, the least our algorithms provide is a different perspective on the internal structure of an argument. We consider Density to be the most promising algorithm as it produces rather consistent result even for vastly different corpora. In future work it may be worth investigating whether the availability of multiple argument graphs of the same source has a positive impact on the understandability of an argument for certain types of users. Another area of improvement is the detection of the correct major claim which could be solved using a binary classifier trained on this task. Such approaches could then in turn be combined with our proposed algorithms to achieve even better results.

References

[1] J. Lawrence, C. Reed, Argument Mining: A Survey, Computational Linguistics 45 (2019) 765–818. URL: https://doi.org/10.1162/coli_a_00364. doi:10.1162/coli_a_00364.

[2] C. Stab, I. Gurevych, Recognizing insufficiently supported arguments in argumentative essays, in: M. Lapata, P. Blunsom, A. Koller (Eds.), Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics, EACL 2017, Valencia, Spain, April 3-7, 2017, Volume 1: Long Papers, Association for Computational Linguistics, 2017, pp. 980–990. URL: https://doi.org/10.18653/v1/e17-1092. doi:10.18653/v1/e17-1092.

[3] I. Persing, V. Ng, End-to-end argumentation mining in student essays, in: K. Knight, A. Nenkova, O. Rambow (Eds.), NAACL HLT 2016, The 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language

Technologies, San Diego California, USA, June 12-17, 2016, The Association for Computational Linguistics, 2016, pp. 1384–1394. URL: https://doi.org/10.18653/v1/n16-1164. doi:10.18653/v1/n16-1164.

[4] M. Lenz, P. Sahitaj, S. Kallenberg, C. Coors, L. Dumani, R. Schenkel, R. Bergmann, Towards an argument mining pipeline transforming texts to argument graphs, in: H. Prakken, S. Bistarelli, F. Santini, C. Taticchi (Eds.), Computational Models of Argument - Proceedings of COMMA 2020, Perugia, Italy, September 4-11, 2020, volume 326 of *Frontiers in Artificial Intelligence and Applications*, IOS Press, 2020, pp. 263–270. URL: https://doi.org/10.3233/FAIA200510. doi:10.3233/FAIA200510.

[5] H. Wachsmuth, M. Potthast, K. A. Khatib, Y. Ajjour, J. Puschmann, J. Qu, J. Dorsch, V. Morari, J. Bevendorff, B. Stein, Building an argument search engine for the web, in: I. Habernal, I. Gurevych, K. D. Ashley, C. Cardie, N. L. Green, D. J. Litman, G. Petasis, C. Reed, N. Slonim, V. R. Walker (Eds.), Proceedings of the 4th Workshop on Argument Mining, ArgMining@EMNLP 2017, Copenhagen, Denmark, September 8, 2017, Association for Computational Linguistics, 2017, pp. 49–59. URL: https://doi.org/10.18653/v1/w17-5106. doi:10.18653/v1/w17-5106.

[6] C. Stab, J. Daxenberger, C. Stahlhut, T. Miller, B. Schiller, C. Tauchmann, S. Eger, I. Gurevych, Argumentext: Searching for arguments in heterogeneous sources, in: Y. Liu, T. Paek, M. S. Patwardhan (Eds.), Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics, NAACL-HLT 2018, New Orleans, Louisiana, USA, June 2-4, 2018, Demonstrations, Association for Computational Linguistics, 2018, pp. 21–25. URL: https://doi.org/10.18653/v1/n18-5005. doi:10.18653/v1/n18-5005.

[7] B. James, Freeman. dialectics and the macrostructure of arguments: A theory of argument structure, 1991.

[8] I. Habernal, I. Gurevych, Argumentation mining in user-generated web discourse, CoRR abs/1601.02403 (2016). URL: http://arxiv.org/abs/1601.02403. arXiv:1601.02403.

[9] I. Persing, V. Ng, Unsupervised argumentation mining in student essays, in: N. Calzolari, F. Béchet, P. Blache, K. Choukri, C. Cieri, T. Declerck, S. Goggi, H. Isahara, B. Maegaard, J. Mariani, H. Mazo, A. Moreno, J. Odijk, S. Piperidis (Eds.), Proceedings of The 12th Language Resources and Evaluation Conference, LREC 2020, Marseille, France, May 11-16, 2020, European Language Resources Association, 2020, pp. 6795–6803. URL: https://aclanthology.org/2020.lrec-1.839/.

[10] C. Stab, I. Gurevych, Parsing argumentation structures in persuasive essays, Comput. Linguistics 43 (2017) 619–659. URL: https://doi.org/10.1162/COLI_a_00295. doi:10.1162/COLI_a_00295.

[11] D. Gemechu, C. Reed, Decompositional argument mining: A general purpose approach for argument graph construction, in: A. Korhonen, D. R. Traum, L. Màrquez (Eds.), Proceedings of the 57th Conference of the Association for Computational Linguistics, ACL 2019, Florence, Italy, July 28- August 2, 2019, Volume 1: Long Papers, Association for Computational Linguistics, 2019, pp. 516–526. URL: https://doi.org/10.18653/v1/p19-1049. doi:10.18653/v1/p19-1049.

[12] R. Schaefer, M. Stede, Argument Mining on Twitter: A survey, it - Information Technology 63 (2021) 45–58. URL: https://www.degruyter.com/document/doi/10.1515/itit-2020-0053/html. doi:10.1515/itit-2020-0053.

[13] R. J. G. B. Campello, D. Moulavi, J. Sander, Density-Based Clustering Based on Hierarchical Density Estimates, in: J. Pei, V. S. Tseng, L. Cao, H. Motoda, G. Xu (Eds.), Advances in Knowledge Discovery and Data Mining, Lecture Notes in Computer Science, Springer, 2013, pp. 160–172. doi:10.1007/978-3-642-37456-2_14.

[14] L. McInnes, J. Healy, S. Astels, Hdbscan: Hierarchical density based clustering, The Journal of Open Source Software 2 (2017) 205. URL: http://joss.theoj.org/papers/10.21105/joss.00205. doi:10.21105/joss.00205.

[15] D. R. Smith, The design of divide and conquer algorithms, Science of Computer Programming 5 (1985) 37–58. URL: https://www.sciencedirect.com/science/article/pii/0167642385900036. doi:10.1016/0167-6423(85)90003-6.

[16] J. MacQueen, Some methods for classification and analysis of multivariate observations, in: Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability, volume 1, 1967, pp. 281–297. URL: https://projecteuclid.org/ebooks/berkeley-symposium-on-mathematical-statistics-and-probability/Some-methods-for-classification-and-analysis-of-multivariate-observations/chapter/Some-methods-for-classification-and-analysis-of-multivariate-observations/bsmsp/1200512992?tab=ChapterArticleLink.

[17] D. Walton, C. Reed, F. Macagno, Argumentation schemes, Cambridge University Press, 2008.

[18] P. J. Rousseeuw, Silhouettes: A graphical aid to the interpretation and validation of cluster analysis, Journal of Computational and Applied Mathematics 20 (1987) 53–65. URL: https://www.sciencedirect.com/science/article/pii/0377042787901257. doi:10.1016/0377-0427(87)90125-7.

[19] A. Peldszus, M. Stede, Joint prediction in mst-style discourse parsing for argumentation mining, in: L. Màrquez, C. Callison-Burch, J. Su, D. Pighin, Y. Marton (Eds.), Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing, EMNLP 2015, Lisbon, Portugal, September 17-21, 2015, The Association for Computational Linguistics, 2015, pp. 938–948. URL: https://doi.org/10.18653/v1/d15-1110. doi:10.18653/v1/d15-1110.

[20] P. Jaccard, The Distribution of the Flora in the Alpine Zone, New Phytologist 11 (1912) 37–50. URL: https://nph.onlinelibrary.wiley.com/doi/abs/10.1111/j.1469-8137.1912.tb05611.x. doi:10.1111/j.1469-8137.1912.tb05611.x.

[21] J. Nešetřil, P. Ossona de Mendez, Bounded Height Trees and Tree-Depth, in: J. Nešetřil, P. Ossona de Mendez (Eds.), Sparsity: Graphs, Structures, and Algorithms, Algorithms and Combinatorics, Springer, 2012, pp. 115–144. URL: https://doi.org/10.1007/978-3-642-27875-4_6. doi:10.1007/978-3-642-27875-4_6.

[22] M. Lenz, R. Bergmann, User-Centric Argument Mining with ArgueMapper and Arguebuf, in: Computational Models of Argument, volume 353 of *Frontiers in Artificial Intelligence and Applications*, IOS Press, Cardiff, Wales, 2022, pp. 367–368. URL: https://ebooks.iospress.nl/doi/10.3233/FAIA220176. doi:10.3233/FAIA220176.

[23] L. Dumani, M. Biertz, A. Witry, A.-K. Ludwig, M. Lenz, S. Ollinger, R. Bergmann, R. Schenkel, The ReCAP Corpus: A Corpus of Complex Argument Graphs on German Education Politics, in: 2021 IEEE 15th International Conference on Semantic Computing (ICSC), 2021, pp. 248–255. doi:10.1109/ICSC50631.2021.00083.