Acknowledgements

I would like to express my gratitude to my supervisor, Dr. Bruce McLaren, for his guidance in my work on this thesis, his numerous comments and his great reviews that definitely significantly improved the quality of this work. Special thanks to all researchers working on the Argunaut project, particularly to the pedagogical experts, Rakheli Hever and Maarten De-Laat, who provided me with helpful suggestions and annotated data for the experiments; it was my pleasure to cooperate with all of them!

Last but not least, great thanks to my family and my friends in Saarbrücken and Prague, for their support and tolerance especially in the final phase of the work.

This work was performed in the German Research Center for Artificial Intelligence (Deutsches Forschungszetrum für Künstliche Inteligenz) and as part of ARGUNAUT project that is partially funded by the EC under the 6th Framework Program (IST-2005-027728).

I hereby declare that I wrote the thesis by myself and listed all used sources. I agree with making the thesis publicly available.

Prague, December 14, 2007 Jan Mikšátko

Contents

List of Figures

List of Tables

List of Abbreviations

AE	Cluster type: “Argument + Evaluation”
AI	Artificial Intelligence
CoO	Cluster type: Chain of opposition
CoOp	Cluster type: Clarification of opinion following feedback
CSCL	Computer-Supported Collaborative Learning
DOCE	Detection Of Clusters by Example
ED	Edit Distance
Exeter	The University of Exeter, UK
HUJI	The Hebrew University of Jerusalem, Israel
MI	Moderator’s Interface
ML	Machine Learning
QBE	Query-By-Example
UDE	The University of Duisburg-Essen, Germany

Abstract

Název práce: Analýza a rozeznávání komplexních interakcí v e-diskusích studentů pomocí technik strojového učení

Autor: Jan Mikšátko

Katedra: Katedra teoretické informatiky a matematické logiky

Vedoucí diplomové práce: Dr. Bruce M. McLaren

E-mail vedoucího: bmclaren@cs.cmu.edu

Abstrakt: Visuální diskuse je formou debaty, ve které se příspěvky zapisují do grafických tvarů a propojují se s ostatními na základě jejich vztahů. Pro moderování několika současně probíhajících diskusí je důležité nasměrovat učitele na zajímavé skupiny příspěvků. Pro tento účel jsme navrhli algoritmus, který používá isomorfismus grafů společně s textovou analýzou a klasifikátory strojového učení pro vyhledávání těchto interakcí na základě poskytnutého příkladu. Vhodnost našeho přístupu jsme experimentálně ověřili na reálných diskusích se slibnými počátečními výsledky.

Klíčová slova: umělá inteligence, klastrování, isomorfismus grafů, visuální e-diskuse, počítačem podporované skupinové učení

Title: Using Machine Learning Techniques to Analyze and Recognize Complex Patterns of Student E-Discussions

Author: Jan Mikšátko

Department: Department of Theoretical Computer Science and Mathematical Logic

Supervisor: Dr. Bruce M. McLaren

Supervisor's e-mail address: bmclaren@cs.cmu.edu

Abstract: Visual e-discussion is a form of debate in that contributions are written into graphical shapes and linked to one another according to their relationship. In order to moderate several simultaneous e-discussions effectively, it is important to point teachers to interesting clusters of contributions. We designed an algorithm that uses inexact graph matching along with text analysis and machine learning classifiers for searching for such patterns based on a provided example. The method was evaluated on a dataset of real discussion and demonstrated promising initial results.

Keywords: artificial intelligence, unsupervised clustering, inexact graph matching, visual e-discussion, computer-supported collaborative learning

Chapter 1

Introduction

1.1 Moderation of Visual E-Discussions

One of important trends in Computer-Supported Collaborative Learning (CSCL) is the development and use of networked visual argumentation tools that allow students to work on separate computers and express their ideas, questions and arguments in visual fashion. Such visual e-discussions in the form of argumentation maps provide an effective and efficient way to teach students reasoning and critical thinking. Students make contributions to the online discussion by dragging and dropping shapes with different meanings (e.g. "claim" or "question"), filling them with text containing their contributions to the discussion and linking the shapes to other relevant shapes with labeled links, such as "opposes" or "supports". An example of such an e-discussion in Digalo (www.dunes.gr) collaboration software is shown in Figure 1 (text in the shapes shows only title of the contribution).

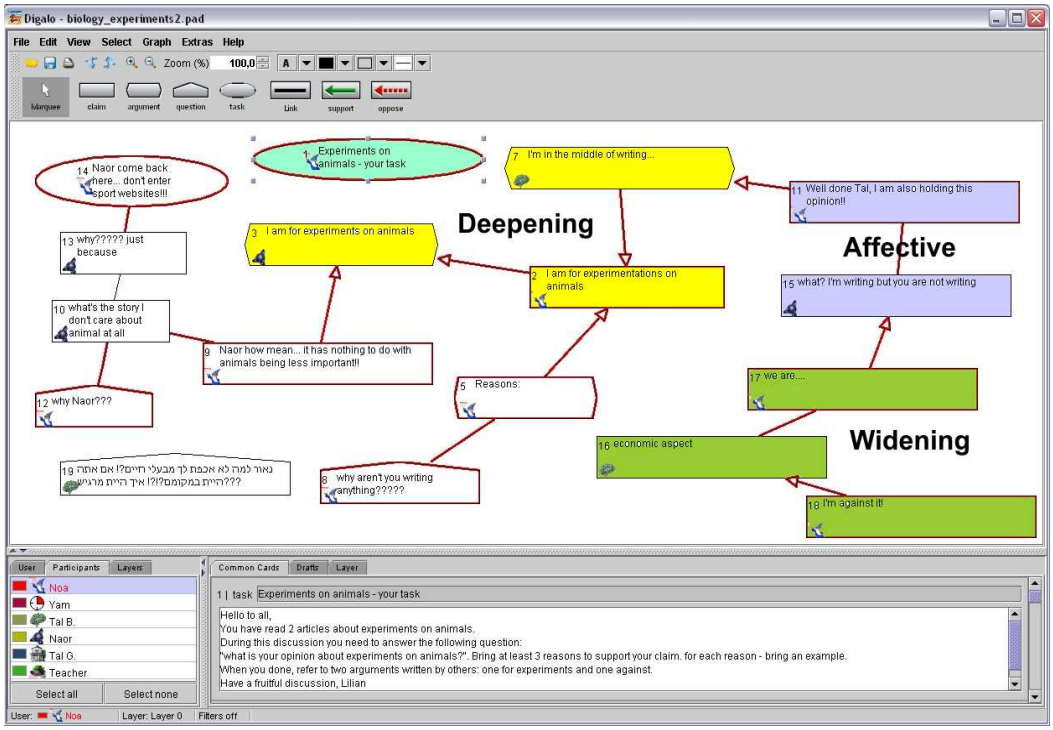

Figure 1: A well-structured discussion in Digalo software with three simple clusters

Although computer-based tools for collaboration, argumentation, and discussion are becoming relatively commonplace in schools [1-3], there is a serious need for software that can help teachers observe, guide, and moderate such e-discussions. For instance, suppose a classroom of students, grouped in small discussion groups of 4 to 6 students, is tasked with discussing and debating a social sciences topic such as "Is it ethical to perform experiments on animals?" using a visual collaboration tool. In a typical classroom of 25 students, this would mean that anywhere from 4 to 7 subgroups of students are engaged in e-discussions at the same time. The teacher in such a classroom obviously cannot monitor and moderate all of these discussions simultaneously without some system (i.e., automated) support. Furthermore, past research suggests that discussion and collaboration tools used by students, on their own with no support or provided structure, such as what might be provided by a teacher or system, do not typically lead to fruitful collaboration [4].

Thus, either a teacher or an automated system component, which can quickly and efficiently identify issues that arise and provide feedback to students, must be present in the e-discussion in order to avoid ineffective or chaotic communication and guide the students toward desirable results. While our long-term goal is to automate support and feedback for the collaborating students – in typical intelligent tutoring fashion [5] – our initial goal is to provide automated support for the teacher, so that he or she can moderate the e-discussions. In particular, in the model we are currently working with, the teacher observes the e-discussion and intervenes if needed, as shown in Figure 2.

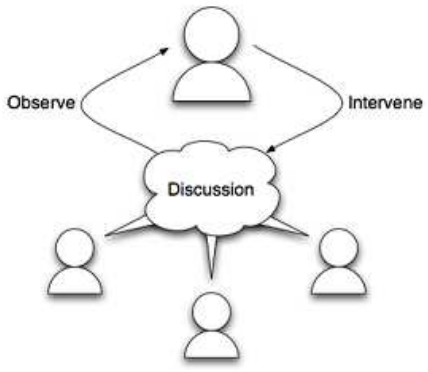

Figure 2: Moderation of e-discussion [6]

In our work on the Argunaut project [8,7], we are exploring ways in which we can support the teacher through the use of machine-learning techniques. The centerpiece of the Argunaut system is the Moderator's Interface (MI) – a software tool that

displays multiple simultaneous e-discussions taking place in the classroom and provides the moderator with feedback regarding important actions in the discussion and points to events requiring a human intervention.

1.2 From Shape Classification to Pattern Detection

In prior work, we used annotated past discussions (collected from several classrooms in Israel and England) and machine learning algorithms to learn interesting individual contributions and paired-contributions, such as off-task behavior and questions followed by answers [9]. These contributions could then be (1) categorized in real-time by a classifier created from the learning cycle and (2) highlighted for the teacher within the MI. This initial effort was relatively successful for some categories of interest – we achieved a Kappa > 0.7 for three categories – but the approach suffers from at least one shortcoming: it focuses on only small parts of each discussion, never accounting for more than two student contributions at a time.

In the present work, we are trying to address the problem of identifying complex interaction patterns in the e-discussions. Such patterns, called *clusters* in the reminder of the paper, are multiple contributions, typically (but not exclusively) made by different students, which capture interesting interactions in the e-discussion. Figure 1 shows an example of a few such clusters. For instance, “Deepening” captures the notion of extending or expanding upon a single original idea; an interaction among several students in which a claim is discussed and developed further through additional questions and reasons. Types of clusters representing interesting interactions are specified and annotated by pedagogical specialists on the Argunaut project, with an eye toward moderating e-discussions. Our primary aim is to provide teachers, the users of the MI of Argunaut [9], with a tool that can point them to interesting clusters in the discussions. A secondary goal is to support the pedagogical specialists in searching off-line for interesting patterns, as they evaluate and data mine past discussions.

Our task is a daunting one, because

(1) we are dealing with highly complex data (i.e., a combination of graph structure and text),

(2) our maps typically have noisy data (e.g., in some classrooms, students are intentionally not provided with guidelines on using the argumentation tools, thus leading to creative and unusual maps),
(3) clusters, even of the same type (e.g. "Deepening"), may vary in size and structure,
(4) cluster types are difficult to precisely specify, and
(5) we have limited source of annotated data, because annotating clusters in real discussions is extremely time-consuming and difficult.

This work describes initial research in which we have tried to use inexact graph matching and unsupervised clustering techniques to identify clusters in real e-discussions. We explored several approaches with respect to development of cluster specifications researched by the pedagogical specialists and goals in the Argunaut project. Two approaches were examined and experimentally evaluated: (1) a method based on unsupervised clustering and (2) an algorithm that employs Query-By-Example (QBE) principle and inexact graph matching. The first approach did not lead to promising initial results, and we decided not to pursue it further (and thus it will only be minimally described in this paper). However, the latter method, referred to as *DOCE* (Detection of Clusters by Example), demonstrated very promising preliminary results on an initial set of annotated maps and thus we decided to pursue it in greater depth. The main advantages of the DOCE algorithm include tolerance to noise and inaccuracies in the discussions, capability to work with imprecise cluster description and limited number of annotations required. It is this latter approach that will be the primary focus of this paper.

1.3 Project Background

Argunaut is an interdisciplinary 3-year research project supported by the 6th Framework Program of the European Community. The project team consists of seven institutions from Germany, the UK, Israel, France and Greece with expertise in AI, pedagogy, CSCL, computer science, and software development. The goal of the project is briefly expressed in its mission statement (citing from Argunaut's website: www.argunaut.org):

> "*The Argunaut consortium was formed with the purpose of providing moderators with a computerized tool and its associated pedagogical*

methodology to support and increase their effectiveness and thereby the quality of the monitored e-discussions."

The Moderator's Interface displays multiple concurrent visual discussions and indicates important events using a series of "Awareness Indicators". There are two categories of Awareness Indicators in Argunaut. "Shallow Loop" indicators are straightforward calculations, based on relatively simple aspects of the discussion, such as counts of the contributions and the shapes used by each student. Figure 3 shows the user interface of the MI displaying a discussion and three Awareness Indicators on the right. Machine-learning classifiers, based on annotated data from previous discussions, were used for "Deep Loop" indicators and provide information such as whether students' contributions are on topic, applied critical reasoning, or were engaged in raising and answering questions. For example, a contribution "Its not nice of human beings to exploit animals for their own needs. I think animals also have rights." would be labeled as on-topic by the Topic Focus Indicator and as applying critical reasoning in an e-discussion about experiments on animals, whereas "I'm bored." is labeled as off- topic and not applying critical reasoning [9].

The Awareness Indicators serve as a summary of important aspects of the e-discussions, helping the moderator decide when and how to intervene in the e-discussion. The MI contains various tools for teacher's intervention ranging from highlighting several shapes to displaying a popup with specific student information. Furthermore, the MI functionality also includes tools for pedagogical researchers to annotate and rate discussion content, or record the discussion flow for later offline analysis.

The Moderator's Interface is a cross-platform tool which currently supports two graphical e-discussion tools: Digalo (developed in DUNES project, www.dunes.gr) and FreeStyler/CoolModes (developed by the Collide group at the University of Duisburg-Essen, www.coolide.info). However, the tool is designed to be platform independent and thus could be applied to other collaborative / argumentation tools.

Research on cluster detection presented in this thesis aims at extending the MI with the following functionality: (1) adding new Deep Loop Awareness Indicators that will highlight important and interesting clusters in the e-discussion for *teachers* and (2) providing a tool for detecting clusters for the purpose of off-line research and annotations for *researchers*.

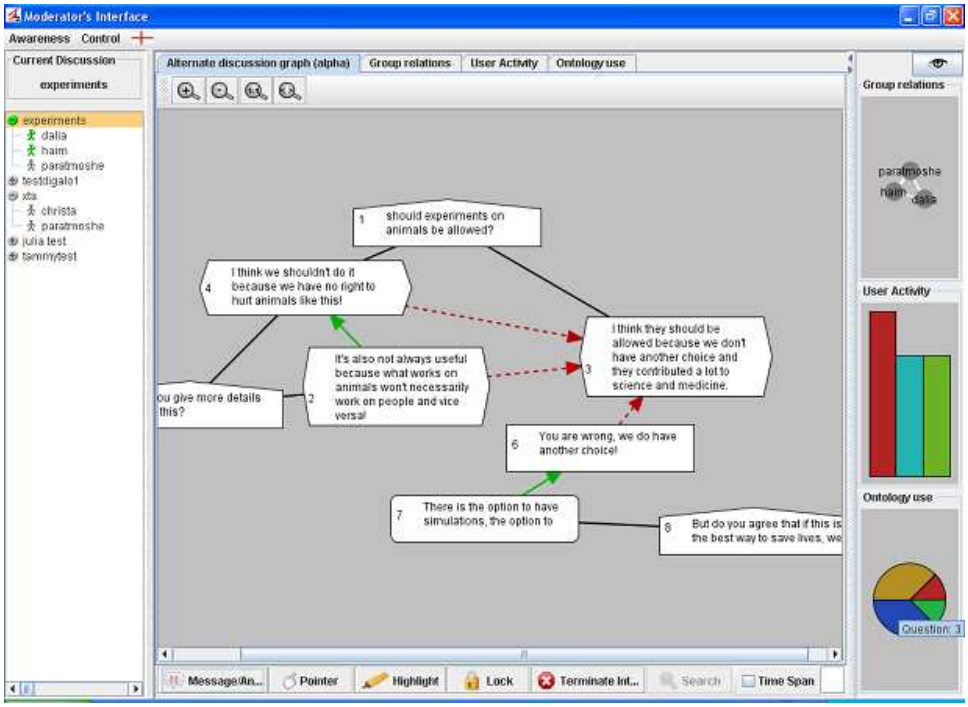

Figure 3: Moderator's Interface

The research reported here was performed at the German Research Center for Artificial Intelligence (DFKI, Germany) – the Argunaut project partner responsible for the AI module and deep loop analysis. There was, however, a close cooperation with the pedagogical specialists on the project, The University of Exeter, UK (denoted as Exeter throughout the document) and The Hebrew University of Jerusalem, Israel (denoted as HUJI), and the technical partners who have developed the Moderator's Interface - The University of Duisburg-Essen, Germany (denoted as UDE) and Fraunhofer Gesellschaft, IAIS, Germany (denoted as FhG).

This section only briefly introduces the project and features relevant to our research. The interested reader is referred to Argunaut's website (www.argunaut.org) for more detailed information about the overall project, a complete list of publications and software downloads. The most relevant publications from the Argunaut project to date can be summarized as follows: [8] describes the Argunaut project; [9] describes initial research and development of the Deep Loop Awareness Indicators, including the application of machine learning techniques, and [10] describes an early pattern analysis tool developed for the Argunaut project; [12,11] discuss moderation practices from pedagogical point of view.

1.4 Related Work

Analyzing student contributions and assigning them labels is common practice in designing and experimenting with intelligent educational technology. For instance, contributions having a particular set of labels might be used as a cue by a software tutor in responding to students in an e-discussion [14,15,13]. Simple characteristics of contributions such as keywords can be used for determining labels. However, [16] investigated more advanced approach by training classifiers on the *language* of a large corpus of labeled data using machine learning techniques. Sets of attributes were extracted from the text segments based on linguistic analysis (e.g., punctuation, unigrams, bigrams, part of speech bigrams) and were used for learning the classifiers for assigning labels to *single* contributions. Promising results in this line of research led to the development of TagHelper – a tool for text classification [17,16] that is also utilized in our work, in particular to help us analyze the language aspects of e-discussion clusters.

A similar approach has been taken in our project and extended to incorporate the *structure* of the discussion in classifying single and paired contributions [9]. The pedagogical specialists proposed variables and designed a coding scheme for individual and pair-level of contributions. Digalo e-discussions from several classrooms were annotated for these variables on the individual level (e.g. Topic Focus, Critical Reasoning) and on the pair-level of two connected contributions (e.g. Question-Answer, Contribution-Supporting Argument). Contributions were characterized by a combination of text features extracted using TagHelper and structural attributes relevant to the e-discussions, such as shape type, link direction, number of in- and out- links, and whether (for paired contributions) different students made the contributions. Three highly reliable classifiers (with Kappa >0.7) were initially trained and integrated in the Moderator's Interface as Awareness Indicators, and with subsequent annotations and experimentation another three or four classifiers have achieved a good enough performance to be added to the system.

Another example of machine learning applied to moderation of online discussion like slashdot.org is suggested in [18]. The authors utilize with success user ratings of previous contributions in the discussion for training a predictor of rating unseen single contributions.

On the other hand, such *supervised* learning approaches do not scale well to clusters of arbitrary size. Clusters have not only to be classified but also *recognized* in the discussion. In the initial work within Argunaut the granularity of analysis (i.e., single contribution or paired contributions) was fully specified and thus easy to detect in real time. But not so with arbitrarily sized clusters. In addition, getting a sufficient number of annotations for training at the cluster level is a very time consuming and difficult task – much more difficult than annotating single and paired contributions. First, there is the problem of the annotators recognizing the clusters. Additionally, there is the problem of the annotators agreeing on the classification of clusters.

A well-researched problem in the data mining literature is detection of frequently reoccurring patterns. Algorithms like GSP [19] detect in unsupervised manner sequences of items that occur multiple times in the dataset. Such an approach was applied in [20] for identifying common interaction patterns during student software development projects on data from source repository logs or Wiki pages. The resulting patterns were used for recognizing problematic projects that may lead to later problems. This approach was also experimentally evaluated on the Argunaut project in [10]. The authors used a Sawtooth algorithm [21] and designed a tool for mining frequent sequences of actions in the discussions, such as create shape, add link or modify text. The tool was able to detect commonly occurring patterns. However, the number of hits was too large for later processing by human experts. In addition, it was not able to detect all the patterns as occurrences of the same cluster type as the clusters tend to differ in subtle and imprecise ways from one another.

Additionally, the work in [10] applies a rule-based approach, using a syntactic specification of attributes (e.g. shape type or link direction), on sequences of actions. Rules can be automatically generated by specifying a sequence example in the log files or by directly writing them in PROLOG. The method succeeded for simple cluster types that can be precisely described by their attributes. Unfortunately, only a few cluster types can be so well specified. It also turned out that the focus on *sequences* of actions – rather than related actions – led to too many false positives. Matching on a static snapshot of the discussion seems to be more appropriate.

While looking for other datasets of annotated data for testing the DOCE algorithm, we explored and contacted many other projects that have worked with and analyzed discussion or argumentation maps. To our knowledge, there has not been a similar study focusing on cluster detection in visual discussions. However, researchers

working on the LARGO project are planning to research and apply similar cluster detection techniques in their tutoring system for law students [22]. Their goal is detecting clusters that represent, for example, weaknesses or opportunities for reflection in visual diagrams of legal cases.

1.5 Thesis Goals

This thesis describes research in the area of cluster detection within visual e-discussions. Identification of complex interaction patterns in noisy, ill-structured visual discussion graphs is a challenging task, one that is even very difficult for pedagogical experts. *Fully* solving the problem is out of scope for this Master's thesis and is perhaps even an impossible task. But this work represents initial research towards cluster detection in e-discussions.

The main contribution of the work can be summarized as follows:

- Exploration of existing approaches for cluster detection (at least those that had clear relevance to the Argunaut project).
- Design and development of DOCE algorithm, a novel clustering method based on the "Query-by-Example" principle together with an inexact graph-matching algorithm.
- Quantitative evaluation of the DOCE algorithm on real discussion graphs.

1.6 Thesis Organization

This paper is organized as follows. The next section describes properties of our discussion and clusters, and suggests approaches for tackling the cluster detection task. The following section focuses on the main method of interest, the DOCE algorithm. The experimental evaluation of the algorithm is presented in chapter 4. Finally, section 5 discusses future research directions and section 6 concludes the paper.

Chapter 2

Approaches to Cluster Detection

There is no obvious and optimal approach for tackling the Argunaut cluster detection problem. In fact, a method may not even exist that can fully solve the problem. The algorithm design is dependent, at least partially, on cluster definitions that are still under development. Moreover, to our knowledge, there are no prior studies close enough to the domain and goals of our project that can guide the work. Therefore, the research has and will proceed iteratively, starting with simpler methods and easier cluster types, leading to more complex approaches and cluster types, all the while leveraging new cluster definitions as they become available.

This section discusses ideas that were considered at the beginning of the project with respect to (1) very limited cluster description from the specialists (consisting of few examples in the maps), (2) complexity of the problem, (3) future development of the cluster description.

Before discussing the approaches, we first describe characteristics of our discussion maps and properties of clusters specified by the pedagogical specialists on this project.

2.1 Discussion Maps

The Argunaut discussions employ a visual representation of arguments, similar to Concept maps and Mind maps. Such a visual approach has been applied to various domains ranging from Philosophy to Law to Education. The iLogos Webpage of argumentation software [23] nicely summarizes projects and tools relevant to argument, mind and concept mapping.

Visual discussions, such as those used in our project, differ from chat-like text discussions in several ways. First, each contribution consists of both a title and text content, rather than text-only as in a chat. Second, contributions in a visual discussion are written into different shapes having different semantics, for example a “question,” a “claim” or an “explanation.” Third, the contributions may be linked to one another

and express relationships between pairs of messages such as “supporting” or “opposing”. Thus, the structure of a visual discussion can be described by a graph structure that may contain cycles and unconnected vertices. Finally, our discussions have a different chronological property than chats since contributions can be modified over time, and it is not always clear from looking at a visual discussion which contributions were made before others. Also, new links may be added between contributions long after the contributions have been initially created.

Data in our project is stored as sequence of actions. The action log is represented in XML format that is common for both of the discussion tools used on the project (i.e., Digalo, Free Styler) and is referred to as the *common format.* In our experiments, we use another XML format that can be extracted from the common format and that represents a snapshot of the discussion at a single point in time (e.g. the final state of the discussion). The time property is captured via creation and modification timestamps associated with shapes and links.

Dataset	Origin	Students	Example of Teacher's assignment
1st HUJI set	Ziv School, Israel	8th and 9th grade	On the one hand, the development of nuclear energy has many economical and military advantages. On the other hand, it has many ecological disadvantages. After researching and reading on this topic, you're asked to advise the country of "Urania", which is similar to Israel in its military and economical conditions. The question being asked is, should the country of "Urania" develop nuclear technology for peacefu; purposes (such as creating electricity)?
2nd HUJI set	David Yelin College & Hebrew University, Israel	graduate and under-graduate	Should the male be separated from the female and the offspring after birth?
3rd Exeter set	School of Education and Lifelong Learning, UK	under-graduate	Is computer supported collaborative learning better for supporting reflection and higher order thinking skills than face to face?
4th HUJI set	Gvanim School, Israel	8th and 9th grade	Read the text 'should Michal go to the party'. After reading the text, answer the question: should Michal go to the party? Base your decision on the texts you've read and the discussion held in the classroom.

Table 1: Datasets of real discussions for experiments

Visual discussion tools are being used in classrooms in several countries. Data from schools in Israel and the UK were collected for the experiments discussed in this thesis. The Hebrew maps were first translated into English. Table 1 summarizes the datasets and describes their origin with samples of discussion topics. Note that students make errors during discussions, for instance, they might forget to link two contributions or choose the wrong shape type. The level of noise introduced by mistakes may vary across the maps. Especially the maps from the Israeli schools contain very noisy and ill-structured data, and thus represent a challenge for our algorithm. This is due to a policy by the teachers of not giving guidelines on using the discussion tools.

Our discussion tools contain *a priori* specified shapes and links, but this can be redefined and extended per discussion. Thus, we defined a simple ontology for remapping shapes and links to a unified scheme across discussions for the experiments. The shape/link types and their mapping to our ontology are shown in Table 2.

Shape Type	
Digalo Type	**Mapped To**
argument	argument
founding	
reason	
claim	claim
idea	
explanation	explanation
comment	information
example	
information	
other	
quotes	
reflection	
problem	question
question	
task	task

Link Type	
Digalo Type	**Mapped To**
link	link
other	
against	opposition
opposition	
for	support
support	

Table 2: Shape/Link types and their remapping

Appendix A contains an argumentation map *biology_experiments3* from the *1st set HUJI* discussing ethical issues regarding experiments on animals. This argument map will be used as an example throughout the document.

2.2 Cluster Definitions

Clusters in our discussions represent interaction patterns that consist of several contributions, with relationships between contributions expressed by links. Clusters may have arbitrary size. The pedagogical specialists on Argunaut specified the cluster types of interest. Our goal is to detect such clusters in new maps. This is a difficult task, both in annotating the example clusters and in designing an appropriate algorithm.

For the shape and paired-shape analysis [9] the experts from HUJI and Exeter annotated lots of contributions. Disagreements were then discussed, an "annotations manual" was created, and a new round of annotations was done. For the second round inter-rater reliability was calculated (as Cohen's Kappa). Such an iterative process of coding refinement resulted in a high level of agreement.

On the other hand, defining a coding scheme for clusters is a significantly more challenging task. Questions that must be answered include: What defines the cluster, its structure, text content or both? What guidelines should be used to determine if a contribution belongs to the cluster? Is it possible to reach agreement between annotations of two independent experts, i.e., achieve high inter-rater reliability? Answering these questions proved difficult considering the complexity of clusters and the answers have not yet been found by the pedagogical specialists. The coding manual for clusters is under development and cluster types are still being researched. Therefore, a high inter-rater reliability on annotations among pedagogical researchers cannot be currently achieved. Accordingly, the clusters are not precisely defined and are changing over time.

Furthermore, the HUJI and Exeter pedagogical experts have different perspectives on researching and defining clusters. HUJI specialists are focused on identifying groups of contributions that represent interesting, important or problematic phenomena based on their pedagogical experience. When such phenomena re-occur across discussions, an attempt is made to label and characterize them as a single cluster type. HUJI patterns exemplify how collaborative argumentation occurs, how people create an argument together through discussion. For example, an interesting case is a pattern in which challenges from others can lead to improved group reasoning (e.g. clarifying a position, providing more information, giving reasons). Additionally, HUJI clusters focus on structural properties of the discussion. For

example, the cluster *Clarification of opinion following feedback* is described as follows (citing HUJI):

> *"These are typically small clusters, usually simple 3 shape chains where 2 of the shapes are by the same user. We usually see something person A says, something person B says in response, and then person A replies (either linking to their first contribution or person B's contribution). This reply has some sort of clarification of their opinion, and/or what they tried to express in their other contribution."*

On the other hand, the Exeter researchers use content-based discourse analysis motivated by dialogic theory as described in [12]. They first identify interesting sequences of contributions in the discussions maps and then define labels for them according to the coding scheme. Exeter clusters are more focused on the text content of the contributions. For example, the cluster *Support sequence* is defined as follows (citing Exeter):

> *"This sequence shows purely support without any reasoning. This sequence often shows simple 'I agree' statements to support someone else's opinion"*

The examples above capture the current definitions of clusters at the time of writing. The cluster descriptions from both groups are in Appendix B. As an example, consider Table 3 that contains several clusters in the map *biology_experiments3* (see Appendix A) as annotated by the specialists.

Author	Cluster Type	Occurrences
Exeter	Deepening discussion with multiple opinions	{8,25,32,44,51,57}, {8,25,32,38,58(59)}
Exeter	Support sequence	{8,16}, {22, 31}, {26, 54}
HUJI	Clarification of opinion following feedback	{22,31,33}, {32,44,51}
HUJI	Argument + Evaluation	{38, 50}, {35, 39}, {(38), 58, 61}

Table 3: Clusters annotated by pedagogical experts in the map *biology_experiments3*

2.3 Approaches Considered

Now we briefly discuss approaches that were considered with respect to the initial cluster descriptions provided by pedagogical experts – several annotated examples of very complex clusters from Exeter and several cluster definitions from HUJI based on top-down research approach. HUJI used a different research strategy at the beginning

where hypotheses based on discussion structure were defined first, and then the discussions were searched in order to verify if such clusters exist. This overview is not exhaustive and more approaches may be investigated in the future as the research on the pedagogy side progresses.

2.3.1 Unsupervised Clustering

Clustering algorithms assign objects (e.g. contributions in the discussion) into different groups according to their similarity that is expressed by a distance measure. There exist plenty of algorithms and the approaches differ in the way of utilizing the distance and grouping the objects into the clusters [24,25].

The resulting clusters represent sets of similar contributions. The motivation of evaluating this method is based on an assumption, drawn by the pedagogical specialists, that contributions in the same cluster should demonstrate some degree of similarity between one another. Moreover, the methods work in unsupervised manner and thus no annotations of clusters or cluster descriptions are required. Although the method cannot label the clusters and may produce false positives, it may serve as a tool in the initial research for detecting candidates for annotation process by pedagogical experts. Another possible application is conceptual clustering – an extension to the basic idea that can extract rules describing detected clusters (e.g. [26]). Such rules may be utilized by another algorithms and further research by pedagogical specialists.

2.3.2 Pattern Mining

Pattern mining is another unsupervised method that detects frequently occurring patterns in the dataset. Basic methods, such as those discussed in “Related Work” section, can be extended to mining frequently occurring (similar) subgraphs [25]. Furthermore, the approach can be provided with expert knowledge in form of constrains in order to limit the number of mined clusters and guide the algorithm towards expected results.

This method is suitable for detecting commonly occurring clusters without labeling them. Pattern mining, like the unsupervised clustering method, does not require annotations and expert knowledge (if no constraints are used). The main drawback is complexity of the approach if extended to graph structures and limited set of clusters that might be detected (only the frequently occurring ones).

2.3.3 Rule-based Approach

This approach is focused on clusters that can be well described by a set of rules that is specified by an expert in a language such as Prolog or JESS. Clusters satisfying the rule set are then returned by a rule-based matching engine. This method was experimentally evaluated on the Argunaut project on sequences of actions in [10] as briefly mentioned the "Related Work" section. The algorithm used only structural properties of the discussion and performed well on datasets without noise and cluster types where the structure is the main cue.

In order to match a wider range of clusters and tolerate noise in the discussions, a relaxed rule language with additional features from text analysis of contributions, for instance, must be defined. Furthermore, the set of rules must be extracted from cluster examples, as most of the cluster types cannot typically be rigorously specified in advance. However, these requirements significantly increase the complexity of the algorithm.

2.3.4 Detection of Cluster by Example (DOCE)

DOCE uses a similar idea to the previous approach, but does not require exact cluster descriptions. An example of a cluster from a discussion map is used as a query and similar clusters across discussions are matched and found using this method. The similarity on content level is computed like in the unsupervised clustering by a distance measure based on shape/link attributes and text features extracted from text analysis of contributions. The graph structure of the query cluster and candidates in the map is compared by an inexact graph matching algorithm.

The advantage of this method is that it requires very few annotations and can target a wide range of cluster types. This method is described in detail in Chapter 3 as it is the primary focus of this work.

2.3.5 Supervised Classification

This approach aims at applying similar concepts as in shape and pair-shape analysis [9]. Concept of the cluster is learned from annotated examples by a machine learning (ML) technique. The input of the classifier is a cluster represented by its content features, for instance shape type or attributes from the text analysis, and by its structural description, for example, the graph structure can be approximated by a degree histogram. In comparison to the shape and pair-shape level, a two-step

algorithm must be applied. In the first step, cluster candidates are generated (e.g. by enumerating clusters of expected size in some way or by an unsupervised clustering method), and, in the next step, candidates are filtered and classified by an ML classifier.

This approach holds promise for being successful for all types of clusters including those that cannot be well described by the specialists. On the other hand, it requires a large number of annotations for the training process and the candidate generation step is nontrivial.

2.4 Choice of Approach

Obviously, given the limits of time and resources it was not possible to investigate and experimentally evaluate all of the above methods. Besides the effort required to implement and test each method, we had to consider the impact of each approach on the pedagogical experts (i.e., how much analysis and annotation work would be required). Furthermore, many of the approaches confronted us with a chicken-and-egg problem in which the algorithm relied on cluster definitions and cluster definitions relied on a software tool to help specify the definitions. For instance, the rule-based method demands a precise definition of cluster types yet the pedagogical experts needed a tool to search the maps and help them provide rule definitions.

Thus, we decided to proceed iteratively, starting with unsupervised techniques requiring no annotations, experimenting next with semi-supervised approaches like DOCE, and then, optionally, testing supervised methods requiring many annotations from the specialists.

In the first iteration, we decided to experimentally evaluate unsupervised clustering, as we believed that the method could lead to a tool for obtaining candidates for annotations. Pattern mining was not chosen since the clusters defined by the pedagogical experts do not necessarily have to frequently occur. Furthermore, this approach was evaluated on the project in [10], although on sequences of actions, with less than impressive results. Thus, the authors decided to not pursue this approach further.

The unsupervised cluster algorithm that we designed and evaluated is described very briefly in the following paragraph because detailed explanation is out-of-scope of this paper. However, the design of the algorithm, implementation and results can be

found on the enclosed CD. Furthermore, the DOCE algorithm (see chapter 3) uses very similar feature representation and distance function.

The unsupervised algorithm was based on the agglomerative hierarchical clustering method [24]. Initially, each instance represents one cluster; next, in each iteration of the algorithm, two clusters are merged by considering two closets instances from both clusters (single linkage criterion). The algorithm repeats this step until all instances are in a single cluster. The data instance, i.e. a contribution in the discussion map, was represented by a feature vector containing attributes from the TagHelper text analysis [16], attributes from the discussion map (e.g. shape/link type), annotations from the shape/pair-shape level analysis [9] and information about the graph structure (e.g. number of in/out links). The distance function is defined as weighted sum of similarity functions relevant to the different types of attributes in the feature vector. Furthermore, the distance function penalizes contributions that are not connected by a link. The data preprocessing was done in Yale [27] and the algorithm was implemented in Matlab.

We qualitatively evaluated the method on a small set of maps from the 1st HUJI dataset. The unsupervised clustering was able to detect meaningful clusters, but its results were not promising enough (more than 50% of detected clusters were false positives) and thus we abandoned this method. Furthermore, the method cannot detect overlapping cluster and label the clusters.

In the next step, a semi-supervised approach – DOCE – was researched with promising results. The rule-based method was not selected because of its focus on well-described clusters (the difficult-to-describe cluster types dominate in the current set).

The reasons for choosing the DOCE approach with respect to the current state of cluster definitions may be summarized as follows:

- The algorithm does not require precisely defined clusters; instead, it employs an intuitive approach in which cluster *examples* are provided.
- Only few annotations are required (as examples for queries) in contrary to supervised methods. Furthermore, it provides a tool for collecting the annotation. Thus, the method solves the chicken-and-egg problem.
- It can detect clusters based on their structural and content features, important to the goals of the Argunaut project.

- It is noise tolerant as it looks for similar, not exactly the same, clusters.
- Additional expert knowledge (e.g. “prefer text content of contributions” or “strictly preserve structure of the cluster”) can be easily incorporated into the algorithm by customizing its parameters.

Finally, we hope that in the future we can pursue a supervised learning approach if enough examples are obtained and annotated, for instance by using the DOCE tool. For example, a trained classifier could be used *per se* on set of generated candidate clusters or integrated as a filter for restricting results produced by the DOCE method.

Chapter 3

Detection of Clusters by Example

This section describes in detail the DOCE algorithm that employs inexact graph matching technique for detecting clusters in discussion maps that are similar to the provided example cluster. The clusters are compared with the model according to both, their structural and content properties.

3.1 Detection of Clusters by Example Concept

The concept is based on the Query By Example (QBE) technique was originally devised in the mid 70s and applied to databases as a query method. The idea behind the QBE is to search for similar files or documents based on an input example: a text string, a document, or visual table example [28]. More recently, this idea has been applied to search engines, for example consider Google. One can specify a URL of a web page and search directly for similar pages or click on a "Similar Pages" link next to a search result and retrieve pages with similar content. The AI subfield of case-based reasoning [29] is another research area in which examples (i.e., cases) are used to search for similar instances in a repository of data (i.e., a "case base")

We employed a similar idea for cluster detection in discussion maps. The way it works is as follows. A moderator or researcher selects a cluster (e.g. connected individual contributions) in an existing discussion that exemplifies an interesting pattern. The example cluster (also called a "model graph" in the following text) is then used as a search query for similar clusters across other discussion maps (called "input graphs"). The output of the algorithm is a list of matching clusters in the discussion map(s), sorted according to a similarity rating. The general idea of this approach is sketched in Figure 4.

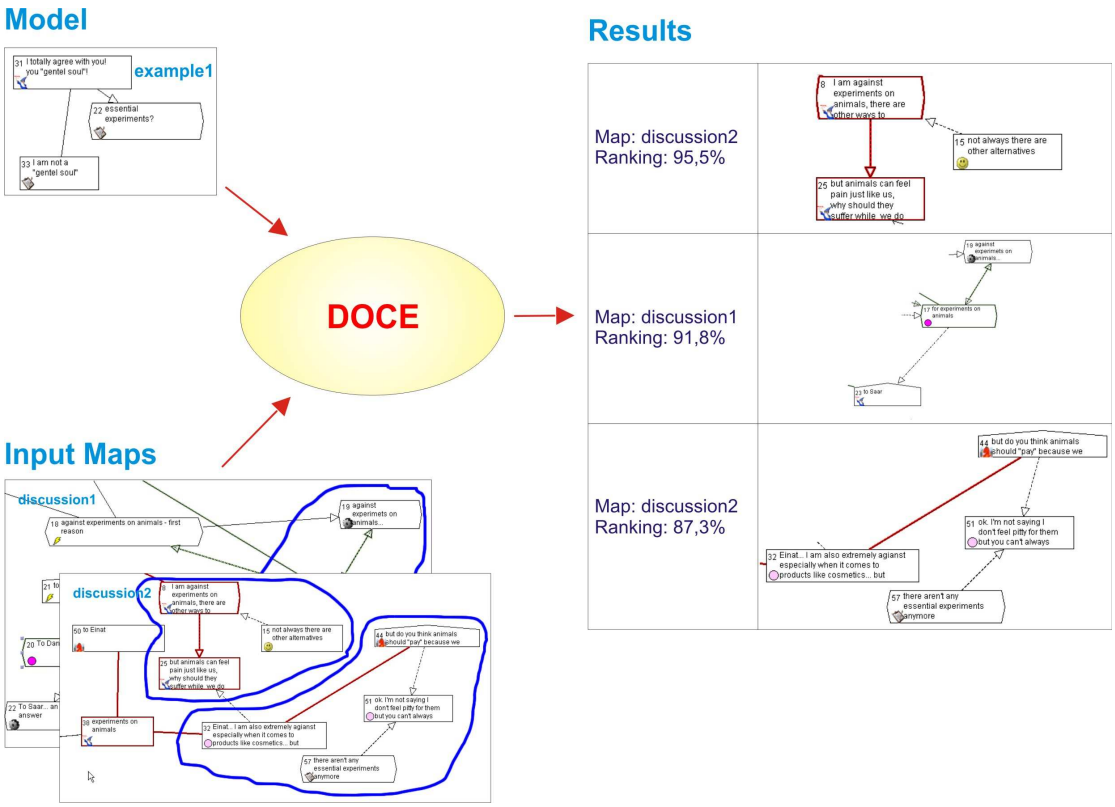

Figure 4: General idea of the DOCE algorithm

DOCE has the advantage of cognitive plausibility and is intuitive. People are typically better at searching for something that “looks” like an example they already have or know about, versus defining what they are interested in words. A further advantage is that DOCE does not require many annotations, as in a typical machine learning approach, but instead provides a tool for collecting and verifying annotations. Only a small set of annotations is required for the evaluation of the algorithm.

DOCE can be used both as an online, automated search mechanism and as an offline research tool. It can be used as a “live” classifier of clusters – characteristic example(s) representing a cluster of a particular type are stored and used later as queries for automated cluster detection. Or, it can be used as a research tool for obtaining clusters and annotating them in the first place. The resulting annotations can then, for example, be used in training a supervised classifier, as was done for shape/pair-shape level analysis [9].

3.2 Graph Matching

Our goal is matching a *model graph* (an example of a cluster) to similar clusters in *input* graphs (discussion maps) A common way of matching two objects and evaluating similarity is extracting/creating a feature vector for each of the objects and measuring the distance between their vectors. In our problem, we are dealing with structured data, consisting of text, relationships between contributions, and additional attributes, that cannot be modeled with a typical a low-dimensional feature vector. Graphs, in particular *attributed graphs*, are powerful and universal representations of complex objects and various similarity measures have been proposed for them.

Definition 1: An *attributed graph G(V, E, ν, ε)* is a graph *G(V, E)* whose vertices have associated a feature vector of *n* features given by function assignment $\upsilon : V \to D_1 \times D_2 \times ... \times D_n$ (e.g. shape type, text length, features extracted from the text) and whose edges have associated a feature vector of *m* features given by function assignment $\varepsilon : E \to D_1' \times D_2' \times ... \times D_m'$ (e.g. link direction, link type). The sets D_i, D_j' are domains for each feature.

The process of finding a correspondence between vertices and edges of two attributed graphs that satisfy some similarity constraints is referred as *graph matching*. In our case we are interested in a generalized and more complex version of this problem – *subgraph matching* – as our input graph is usually much larger than the model graph. Our goal is to find all subgraphs similar to the model in the input graph.

Graph matching algorithms have been used for various purposes, such as computer vision [30], pattern recognition [32,31], bioinformatics [33,34], retrieving relevant principles from ethics cases [35] and web search [36]. For instance, in [32] inexact subgraph matching is used for recognition of engineering symbols in engineering drawings. A symbol is represented as a labeled graph and matched against an input graph – an engineering drawing. [30] describes an application to matching images of natural scenes. A drawing of a known scene is sketched and used as a model graph. Color and texture segmentation algorithms are applied to images of natural scenes and an input graph is extracted. A graph matching algorithm is then used for matching the known scene with the right photo. [35] applies inexact graph matching in the domain of ethics. Engineering ethics cases and principles are modeled

in a stylized language. An undecided case is then matched against past cases and codes using an inexact graph matching technique and provides a human with suggestions in deciding the current case.

A large number of algorithms have been proposed over the last 30 years or so; however, the algorithms have typically been application dependent and vary significantly in their properties, such as restrictions on graph type, exact or inexact matching, computational complexity, support of attributed graphs or possibility of matching subgraphs. A recent paper [37] provides a taxonomy and brief overview of the graph matching algorithms for computer vision based on 160 papers.

3.2.1 Taxonomy of Graph Matching Algorithms

There are two main categories of matching algorithms – *exact* and *inexact*. Exact matching, called *graph isomorphism*, is informally a bijective mapping between vertices of two graphs that strictly preserves structure. The original algorithm was published in [38]. Exact methods are typically based on exhaustive search of possible mappings between vertices of two graphs. In our case, discussion maps are affected by noise (e.g. students make mistakes when linking contributions) and graph structure may vary across examples of the same cluster type. Thus a method that can match similar graphs is required, more specifically an algorithm that can inexactly match subgraphs in the input graph. This can be done by introducing an error model into the matching process and such algorithms are often referred as *error-correcting subgraph isomorphism*.

Inexact matching algorithms can be further split into two categories: *optimal* – guaranteed to find the best solution (according to a matching cost function) with exponential execution time in the worst case; and *approximate* – not guaranteed to find the optimal solution, but these are usually polynomial bounded algorithms.

Computational complexity is the main disadvantage of graph matching algorithms. Graph isomorphism is still an undecided problem whether it belongs to the *P* or *NP* class [39]. However, subgraph isomorphism is well known to be *NP*-Complete [39] and thus error-correcting subgraph isomorphism has the same complexity. In other words, no algorithm fully solving the subgraph matching problem in polynomial time can be designed. The effect of exponential running time in the worst case can be reduced in various ways. Restricted classes of graphs such as trees can be (inexactly) matched in polynomial time [40]. Many approximate

algorithms belonging to *P* class have been proposed, although they may fail to find an optimal solution. Various application-dependent heuristics can be designed to avoid the exponential time in the average case. Past research in this domain has shown that the subgraph matching methods with heuristics behave reasonably well on many real problems [31,41,37].

3.2.2 Properties of our Problem

Properties of our matching problem are briefly summarized in Figure 5. An ideal solution must fulfill the following requirements:

(1) Our discussion graphs are generic graphs that may contain cycles and consist of separate components (a graph consisting of subgraphs that are not connected to each other by edges). Some of our cluster examples are unconnected as well as shapes in the discussion may be unlinked (students sometimes forget to link the shapes).

(2) Occurrences of clusters of the same type do not necessarily have the same graph structure (e.g. see Table 3).

(3) The discussion maps are noisy and contain mistakes (e.g. students choose wrong link direction, wrong shape or forget the link at all).

(4) We are interested in detecting all occurrences similar to the model graph, i.e. we must identify all similar subgraphs in the map. This property moves our problem to the *NP*-Complete class.

(5) The pedagogical researchers would like to query several (or many) discussions at once and, on the other hand, several different model graphs representing the same cluster type might be used as a query. Therefore, *many-to-many* graph matching is required.

(6) We do not want to match graphs based only on structural similarity, but also consider the content of contributions, such as their text, the users involved in the cluster and shape type. The method has to support attributed graphs and utilize the notion of similarity on vertices and edges.

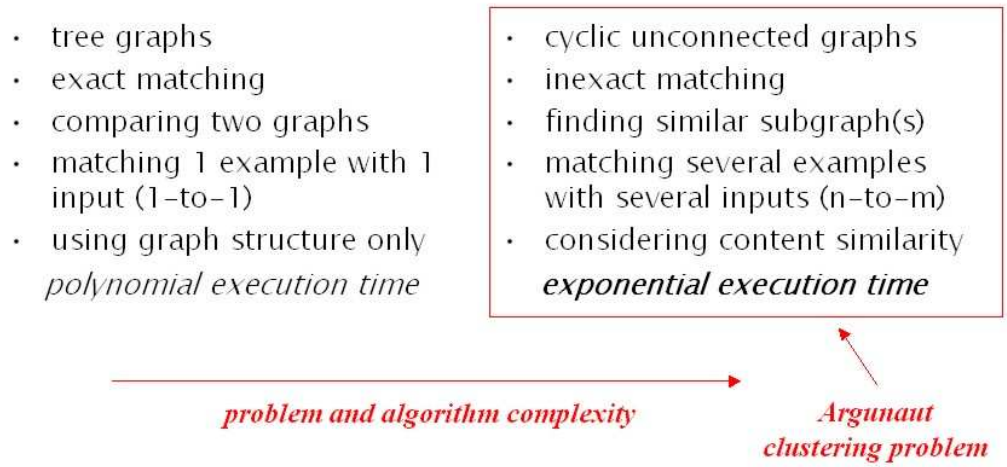

Figure 5: Properties of our problem

To our knowledge, there is no algorithm that fully satisfies such a set of requirements. Furthermore, as mentioned above, our matching problem is NP-Complete. However, key questions are: is the exponential execution time in the worst case a practical problem? Or is it strictly a theoretical problem? More specifically, can we make simplifications to our problem in order to make it tractable?

Table 4 summarizes statistical properties of the *1st HUJI set* of real maps and clusters annotated in the maps for the experiments. Ideally, we would like to simplify the problem to a solution with polynomial complexity, such as to tree graphs. Unfortunately, almost 50% of our input graphs are cyclic and more than 75% contain unconnected vertices and therefore restricting graph structure to trees or limiting the algorithm only to connected graphs is not feasible. However, the relatively small size of the models (on average 3,5 vertices with a maximum of 9 vertices) that are of interest to our pedagogical experts, as well as the small size of the discussion maps (on average 17 vertices with maximum size of 26 vertices) generated by real students is a very helpful simplification, even if we use techniques that have exponential time in the worst case. Furthermore, 80% of our model graphs have size up to 4 vertices and thus we expect very low execution time for most of the models even if the heuristics do not contribute significantly. The specific choice of the underlying method for our DOCE algorithm is discussed in the next section.

Properties of Input Graphs from 1st HUJI Set				
	Nodes	Edges	Properties of the Graphs	
Min	9.0	6.0	Graphs total:	13
Max	26.0	21.0	Graphs w/ cycles	46%
Avg	16.9	14.5	Unconnected graphs	77%

Properties of HUJI+Exeter Model Graphs in 1st HUJI Set				
	Nodes	Edges	Properties of the Graphs	
Min	1.0	0.0	Graphs total:	51
Max	9.0	9.0	Graphs w/ cycles	10%
Avg	3.5	2.9	Unconnected graphs	4%
Models with size <= 4 nodes:				80%

Table 4: Properties of discussion maps from the *1st HUJI Set* and properties of cluster examples annotated for DOCE experiments

3.2.3 Choice of Inexact Graph Matching Algorithm

There are dozens of variants of *inexact* graph matching methods in the literature. Almost all of these algorithms are highly application dependent. The main approaches are summarized below:

Edit distance (ED) methods are optimal matching techniques. Given two graphs G_1 and G_2, a backtracking algorithm (usually A^*) is used to explore all possible assignments between vertices of graphs G_1 and G_2. In each step, a partial mapping of vertices is extended by adding a new vertex-to-vertex assignment. The edit distance between partially matched graphs is calculated as a minimal sequence of primitive graph operations (such as “add an edge”, “delete an edge”, “delete a vertex”) that are required in order to make the graphs isomorphic. Each primitive graph operation has an associated cost and the costs over all steps of the algorithm is the final matching cost. The original algorithm with a limited set of edit operations was proposed in [42]. The work was extended by allowing more edit operations and adding heuristics for reducing the search space, for instance [43,44] among many others. This method is described in detail in the next section, as our DOCE algorithm employs this approach.

Methods based on Maximum Common Subgraph / Minimum Common Supergraph. A graph G is *maximum common subgraph* if it is a subgraph of both graphs G_1 and G_2 and no such larger subgraph exists (by some measure involving number of vertices and edges). This problem is a *NP*-Complete [39]. Similarly, *minimum common supergraph* is the smallest graph containing graphs G_1 and G_2. An algorithm that can solve one of the problems can be trivially modified into an algorithm solving the second problem [45]. Size of the maximum common subgraph

and minimum common supergraph, respectively, can be used as a distance between model and input graph and such metric is suggested in [45,46]. An algorithm using backtracking with efficient heuristics for detecting a minimum common subgraph is given in [47]. These methods belong to optimal matching techniques.

Iterative methods are approximation methods that use a local perspective on the similarity of two graphs. Similarity of two vertices of two different graphs is calculated based only on similarity of their features and their immediate neighborhood. This is a recursive definition and leads to an algorithm iteratively propagating similarity scores to the neighboring vertices in each step. The idea comes from web page ranking algorithms [48]. [49] provides a detailed overview of the method with theoretical discussion of its properties and its extension to subgraph matching problem.

Topological (statistical) methods are approximation methods that use aggregate measures of the graph structure, such as degree histogram or clustering coefficient, for computing similarity of two graphs. Such methods are suitable for large graphs (e.g. social network graphs). [50] gives an example an algorithm based on degree histograms only.

Probabilistic techniques belong to the class of approximation methods. Each vertex of the first graph is assigned a vector with probabilities of an assignment to each vertex in the second graph. Initial vectors are determined heuristicly based on available information such as node attributes or node connectivity. These vectors are iteratively updated by taking into account vectors of neighborhood vertices until the process converges to a fixed point. For each vertex of the first graph an assignment with maximum probability is chosen. The theoretical foundation of this technique is given in [51] and an example of matching algorithm based on random walks is described in [52].

Other approaches. [53] uses Hopfield neural networks for representing each vertex-to-vertex mapping by a neuron and optimizes the output of the network. Another interesting approach based on edge matching distance is given in [54]. The authors define a cost function for matching edges of two graphs and a polynomial algorithm is used for deriving minimal weight set of matching edges.

Unfortunately, there is no detailed study summarizing the properties and evaluating the performance of the algorithms. We decided to choose the edit distance

approach, by far the most used method and one that has been applied to many real problems (e.g. [30,32,31]). Numerous variants of the algorithm were proposed as well as many heuristics. Its flexibility allows us to incorporate all of the requirements of our problem discussed in the section 3.2.2 (see below). The reasons can be summarized as follows:

- It is an *optimal* method ensuring that the algorithm will not miss a solution if it exists. This is in contrast to approximate methods.
- Supports *inexact subgraph* matching on *generic* graph structures that can be extended to matching all similar subgraphs in the input graph. This matches following requirements of our problem: (1), (2) and (3); and with the extension also the requirement (4).
- Heuristics were proposed in order to tackle the exponential running time in the worst case [31,43,44,41]. Two of the heuristics are employed in our implementation in order to limit the effect of exponential running time.
- Specifying appropriate parameter values can be a problem; the more parameters that are required, the more difficult it is to configure the algorithm optimally. Although the *ED* method is not parameter free, it requires a limited number of *a priori* parameters, in contrast to the other methods, for instance probabilistic algorithms, which typically require many parameters.
- It supports attributed graphs and vertex/edge similarity can be incorporated in the matching process, as required in our problem by item (6).
- The algorithm can be extended for supporting many-to-one matching as was proposed [55,56,41]. This extension is not included in our preliminary implementation, but it would partially solve the requirement (5).

In order to adapt the *ED* approach to our problem, we have customized and extended the method as follows:

- The *ED* method can match a model graph to a single similar subgraph in the input graph. The DOCE algorithm, on the other hand, is designed to find several similar subgraphs in the input graph and return them as a set ordered according to their similarity to the model graph.

- *ED* algorithms on attributed graphs consider the values of attributes assigned to vertices/edges as a label and perform a substitution operation with a fixed cost if a vertex/edge in the model graph has a different label from a vertex/edge in the input graph (i.e. the attribute values differs). The DOCE does not use substitution operations, but calculates similarity of vertices/edges based on their features and such value, representing the content similarity, influences the *A** search in choosing the vertex-to-vertex assignment.

3.3 Edit Distance based Matching Algorithm

The DOCE algorithm is sketched in Figure 6. First, the model and input graphs are parsed from an XML file format that is used by the Moderator's Interface for representing a snapshot of the discussion. Both graphs are preprocessed as follows: (1) an adjacency matrix representing the structure of the graph is constructed; (2) each contribution and link in the discussion graph is characterized by a feature vector that is extracted from the attributes associated with the discussion vertex and edge. The feature vectors are further enriched with additional information from the text analysis performed by TagHelper [16] and the shape (pair-shape level analysis for edges, respectively) analysis done by machine-learning classifiers [9].

In the next step, similarity matrices comparing vertex/edge feature vectors of both discussion graphs are pre-computed. Finally, an *A**-based search algorithm is used find vertex-to-vertex mappings between model and input graph vertices by choosing mappings that maximizes content similarity of vertices and edges (pre-computed in the similarity matrices) and minimizes structural differences of the input graph and matching subgraph (measured by the *ED*). First *n* complete mappings are returned as resulting clusters and sorted by their ranking.

This following sections describe each step of the process in detail.

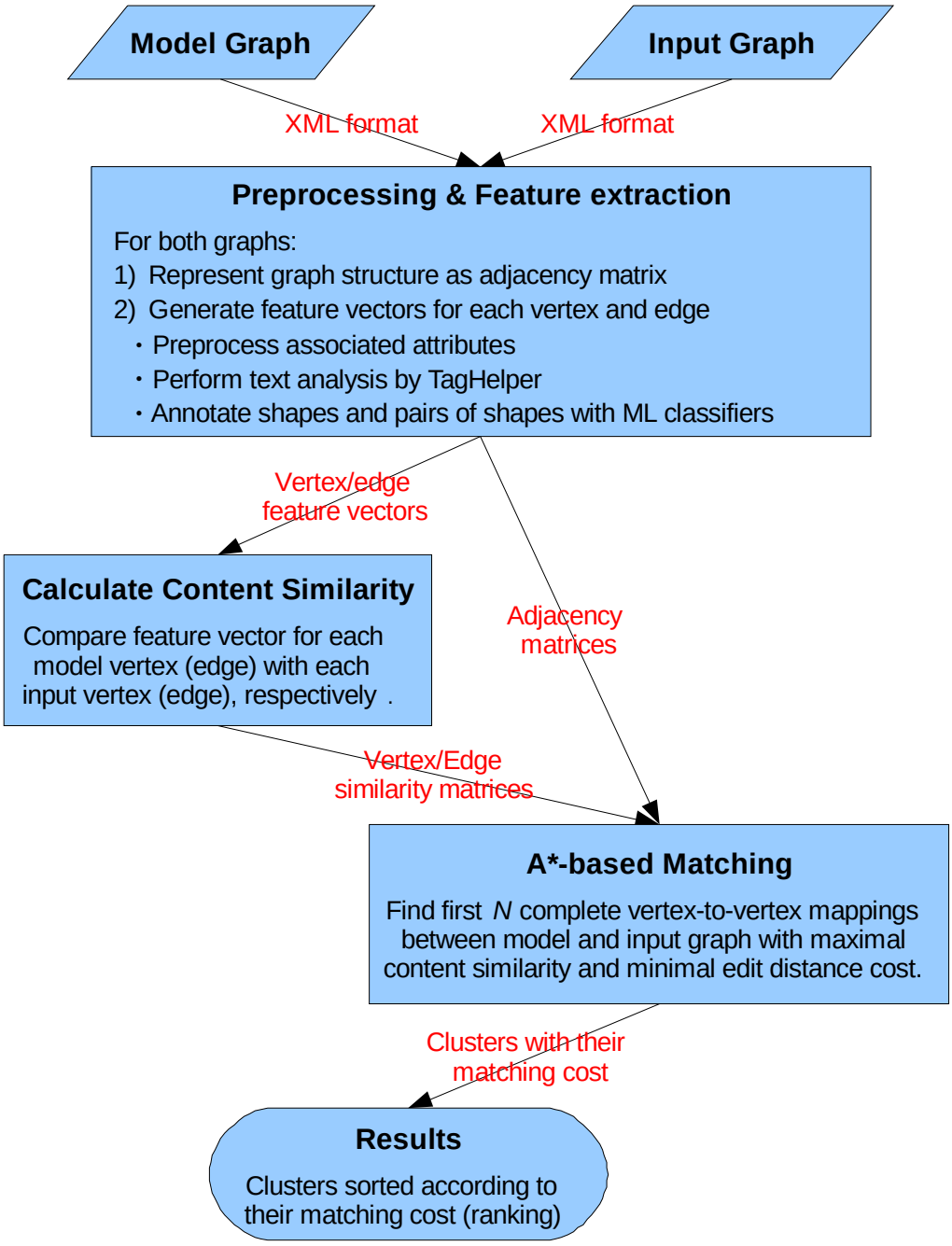

Figure 6: The DOCE algorithm

3.3.1 Preprocessing and Feature Extraction

The input of the algorithm is an input and model graph selected by the user in the MI and passed to the algorithm in *GraphML* XML-based format. The *GraphML* represents the current state of the discussion and can be exported from the discussion tools, the MI or extracted from the *common format* that captures sequence of students' actions (see section 1.3).

Each contribution and link has associated a set of attributes in our discussion maps. The raw attributes are preprocessed into a feature vector of real numbers characterizing a vertex or an edge. Feature vectors are later used for comparing similarity of two vertices or edges between the model and input graph. Feature extraction is a common method for representing objects in order to compare them in unsupervised learning algorithms [25] and we share the same principles and goals in computing similarity of vertices/edges. In the Definition 1 of attributed graph, the feature extraction and feature vector assignment to a vertex or an edge is represented by functions υ, ε, respectively.

Furthermore, we extend the feature vector with additional information that is not included in the discussion map attributes. We leverage output of shape/pair-shape level classifiers that assign contributions and pairs of contributions into categories [9]. An attribute is added to the feature vector per category.

Additionally, text in the contributions is linguistically analyzed by TagHelper tool [16] in order to extract characteristics of the text. Such information is added to the feature vector of vertices. TagHelper extracts the following features from the text:

- *Unigrams and Bigrams* – single words and pairs of words occurring in the text. They characterize the text as keywords. Bigrams can capture the deeper meaning of word phrases, such as “common denominator”.
- *POS bigrams* – part-of-speech bigrams that can express some aspects of syntactical structures.
- *Punctuation* – can be used as simple indicator of question or mood of the author.
- *Contains non-stop words* – value predicting if the text is meaningful or not (e.g. “ok”).
- *Line length* – very simple characteristics that can capture depth of the contribution.
- *Text preprocessing* – includes stemming operation (transforming each word into its lexical root) and removing stop words (removes some distracting features).

The set of features covers all of the information associated with contributions and links in the discussion maps and was designed based on experience with shape/pair-shape level experiments [9] and unsupervised learning experiments (see documents on the enclosed CD for more details).

Attribute		Example of Raw Values		Preprocessing & Feature Extraction	Feature Type
Name	Type	Model Vertex	Input Vertex		
Shape type	categorical	argument	argument	remapped by shape type ontology; conversion to number	positive natural number
Text length	interval	100	52	outliner filtration	natural number in interval <0,MTL>
Text	text	"i think riki and yosi … care of his illness."	"chen say, do you think they should have an abortion?"	concatenation of contribution title and text; TagHelper extraction of features	sets of labels for POS bigrams, puncutiation, unigrams and bigrams; cns is boolean value
Annotations	binary	(1, 1)	(0,1)	produced by ML classifier	vector of {0,1} values

Table 5: Feature Extraction for Vertices

The feature extraction for vertices with examples is summarized in Table 5. Some details are not visible from the table and are explained in the following:

- Text length and text attributes are based on concatenation of the title text and the contribution text.
- Shape type is a categorical variable and its labels may be redefined in the Digalo software. A single ontology is used for unifying the labels across argumentation maps (see Table 2). Nominal values are converted to real values.
- Text length greater than a constant *MTL* is reset to this value. This simple outliner detection turned out to be satisfactory for our data. A more sophisticated statistical-based algorithms [25] could be applied if needed.
- Annotations are a vector of true/false values produced by shape-level ML classifiers. Only categories with high reliability (i.e., classifiers with Kappa

values > 0.65) are included in the features set. The values in the table are categories: (*Topic Focus*, *Chain Of Reasoning*).

- TagHelper is applied separately on the text in each contribution with the following configuration: word stemming, removal of stop words and English dictionary. We let TagHelper produce the following feature sets: POS bigrams, punctuation, unigrams and bigrams (each represents a set of attributes with Boolean values), and a Boolean value "contains non-stop word" (cns).

Attribute		Example of Raw Values		Preprocessing & Feature Extraction	Feature Type
Name	Type	Model Edge	Input Edge		
Link type	categorical	link	opposition	remapping by link type ontology; conversion to number	positive natural number
Link direction	categorical	IN	IN	conversion to number	positive natural number
Time delta	interval	20 s	50 s	-	positive natural number
Same user?	binary	0	0	-	{0,1}
Annotations	binary	(1,0,0,0)	(1,1,0,0)	produced by ML classifier	vector of {0,1}

Table 6: Feature Extraction for Edges

Analogically to the feature extraction for contributions, features summarized in Table 6 are extracted for links in the discussion map. Note the following:

- Link type and link direction are properties of the discussion link. The link type is remapped according to the ontology (see Table 2) similarly to shapes.
- Time delta is the absolute difference between creation times of linked shapes.
- The "Same user?" attribute is an additional feature that characterizes if the two linked contributions were created by the same user or not.
- Annotations represent categories that characterize two linked contributions. These attributes are produced by the pair-level ML classifier. Currently high-reliable classifiers (Kappa > 0.65) used in our experiments are the following: *Contribution-Counter Argument*, *Contribution-Supporting Argument*, *Question-Answer* and *Contribution followed by Question*. Note that the

content of both linked contributions that is used by the classifier cannot be seen from the Table 6.

3.3.2 Content Similarity

When mapping a vertex in the model graph to a vertex in the input graph, the target vertices may differ in their shape or text content from the model vertex. Measuring similarity between pairs of vertices gives the mapping algorithm a guideline for selecting (or preferring) vertices that are more similar to one another, for instance having the same shape or both containing a question. Analogically, the same idea is applied to link similarity, for instance matching link type and/or link direction.

Similarity of two objects can be calculated by the measuring distance between objects by *similarity function*. Such a function typically takes feature vectors characterizing both objects as an argument and returns a value in *[0,1]*, where *0* means that the objects are highly similar and the value ranges toward *1* as they differ [25]. The vertex/edge feature vectors contain variables of different types, and hence different similarity functions are used for each feature subset. The final value is calculated as a weighted sum of partial similarities giving each feature different weight.

Feature	Similarity function	Example of Extracted Values		Result
		Model Vertex	Input Vertex	
Shape type	Equivalence	1	1	0
Text length	Euclidian w/ normalization	100	52	0.192
Text - all	Weighted sum of features:	-	-	0.476
- *POS bigrams*	Jaccard w/ normalization	[NN_IN, ._EOL, VBP_NNS, ...]	[NN_NN, NN_EOL, ...]	0.429
- *punctioation*	Jaccard	[PERIOD]	[QUESTION _MARK]	1
- *unigrams*	Jaccard w/ normalization	[care, ill, yosi, think, riki, ...]	[sai, abort, chen, think]	0.1429
- *bigrams*	Jaccard w/ normalization	[think_riki, yosi_care]	[BOL_chen, chen_sai]	0.1429
- *cns*	Equivalence	1	1	0
Annotations	Simple matching	(1, 1)	(0,1)	0.5

Table 7: Similarity functions for vertices

Similarity functions used for the vertex and edge vectors are summarized in Table 7 and Table 8, respectively. Their definitions can be found in [57,25]. The similarity functions were chosen based on feature type, experience from the unsupervised

learning experiments and recommendations found in [25]. A few explanations are in order:

- *Euclidian distance* is employed for interval variables. The value is normalized for text length to maximum text length constant (*MTL*) and for time delta to maximum time delta (*MTD*).
- *Equivalence* is used for Boolean features and categorical features with one variable (e.g. "Same author?" or shape type).
- *Simple matching distance* is typically used for comparing two Boolean vectors where negative and positive values carry equal information (symmetry), for instance (e.g. annotations).
- *Jaccard distance* is commonly applied to comparing two sets (e.g. unigrams). The value is normalized by an experimentally chosen constant for some TagHelper features instead of union of the feature sets. For such features, the intersection is usually very small and the size of union large. Normalization by a small constant allows proper scaling of the similarity value.
- Text similarity is calculated as a weighted sum of similarity of features set generated by TagHelper with equal weights for each term.
- All calculations produce values in the interval [0,1].

Feature	Similarity function	Example of Extracted Values		Result
		Model Edge	Input Edge	
Link type	Equivalence	1	2	1
Link direction	Equivalence	1	1	0
Time delta	Euclidian w/ normalization	20	50	0.5
Same user?	Equivalence	0	0	0
Annotations	Simple matching	(1,0,0,0)	(1,1,0,0)	0.25

Table 8: Similarity functions for edges

Finally, we can define:

Definition 2: Similarity between two mixed-type vectors *a* and *b* is computed as: $d(a,b) = \sum_{f=1}^{n} w_f d_f(a^f, b^f) / \sum_{f=1}^{n} w_f$, $w_f \in \Re^+$, where *f* iterates over features, $d_f(a^f, b^f)$ is defined similarity function for feature *f* and v^f is projection of feature *f* included in feature vector *v*.

Given attributed graphs $G_1(V_1, E_1, \upsilon_1, \varepsilon_1)$ and $G_2(V_2, E_2, \upsilon_2, \varepsilon_2)$, *vertex similarity* between two vertices $v_1 \in V_1$ and $v_2 \in V_2$ with respect to vertex features and vertex similarity functions defined in the Table 5 and the Table 7 is calculated as: $VertexSim(v_1, v_2) = d(\upsilon_1(v_1), \upsilon_2(v_2))$; and analogically, *edge similarity* between edges $e_1 \in E_1$ and $e_2 \in E_2$ with respect to edge features and edge similarity functions defined in the Table 6 and the Table 8 is then: $EdgeSim(e_1, e_2) = d(\varepsilon_1(e_1), \varepsilon_2(e_2))$.

Similarities for each vertex/edge in the model graph and each vertex/edge in the input graph are calculated prior to executing the *A** search. The computation time is obviously quadratic and the values can be cached for each model-input graph pair.

Choice of weights is discussed in the section 3.3.6. Note that the vertex/edge similarity value will be in the interval *[0,1]* if the selected weights sum to one.

3.3.3 Edit Distance Measure

The edit distance for graphs is motivated by the edit distance for strings [58]. It is an intuitive measure representing a minimum number of operations that must be performed on the model graph in order to transform it into the input graph. The operations may be insertion or deletion of an edge, or insertion or deletion of a vertex. Edit distance expresses structural difference of two graphs

The definitions in this section are customized to our problem and loosely follow the framework given in [56].

Before defining the edit distance of two graphs, we must define the meaning of a subgraph of an attributed graph and subgraph isomorphism on attributed graphs.

Definition 3: Given an attributed graph $G(V, E, \upsilon, \varepsilon)$, a *subgraph* of G is graph $G'(V', E', \upsilon', \varepsilon')$ where $V' \subseteq V$ and $E' = E \cap (V' \times V')$ and υ', ε' are restrictions of functions υ, ε to V', E', respectively. Subgraph will be denoted as $G' \subseteq G$.

Definition 4: Given attributed graphs G_1 and G_2, a bijective function $f : V_1 \to V_2$ is called *isomorphism* from G_1 to G_2 if $\forall (v_1, v_2) \in E_1 :$ $(f(v_1), f(v_2)) \in E_2$ and $\forall (v_1', v_2') \in E_2 : (f^{-1}(v_1'), f^{-1}(v_2')) \in E_1$.

Subgraph isomorphism from G_1 to G_2 is an injective function $f : V_1 \to V_2$ if there exist a subgraph $G \subseteq G_2$ such that *f* is graph isomorphism from G_1 to *G*. Such subgraph will be denoted as G^f.

Informally expressed, a subgraph of an attributed graph *G* is a subgraph with restricted functions assigning attributes to vertices and edges. Subgraph isomorphism is a mapping from G_1 to a subgraph *G* in G_2 that strictly preserves graph structure.

Further, we define the edit operations for altering the model graph. Add/delete edge operations handle missing or extraneous edges in the input graph. For example, a student forgot to link two shapes that are connected in the model graph. Thus, application of edge delete operation to the model graph adjusts it to the input graph. Vertex add/delete operations allow matching to larger/smaller input graph with respect to the model graph. Some input graphs may differ a lot from the model graph and a chain of edit operations must be applied before their structures are isomorphic.

Definition 5: *Graph operation* δ on an attributed graph *G* is one of the following:

- $v \to \$, v \in V$: deletion of a vertex *v* from graph *G*
- $e \to \$, e \in E$: deletion of an edge *e* from graph *G*
- $\$ \to e = (v_1, v_2), v_1, v_2 \in V \,\&\, e \notin E$: insertion of a new edge *e* into the graph *G*
- $\$ \to v, v \notin V$: insertion of a new vertex *v* into graph *G*.

Edited graph is graph *G* after applying one of the edit operations above and will be denoted as $\delta(G)$.

An *edit sequence* is a sequence of edit operations $S = (\delta_1, \delta_2, ..., \delta_n)$ and an edited graph *S(G)* is a graph after applying all of the edit operations in the sequence to graph *G*: $S(G) = \delta_n(...\delta_2(\delta_1(G))...)$.

Each edit operation is assigned a cost in order to penalize structurally different graphs. The edit costs can be chosen a priori based on heuristic knowledge or

adaptively as described in the section 3.3.6. The sum of the edit costs provides a measure of structural distance between the two graphs.

Definition 6: Each edit operation is assigned a nonnegative *edit cost* by the edit cost function: $c : \delta \to \Re^+$. *Total edit cost* of transformation of an attributed graph *G* into *S(G)* using edit sequence $S = (\delta_1, \delta_2, ..., \delta_n)$ is: $edc(S) = \sum_{i=1}^{n} c(\delta_i)$.

Notice that the previous definitions do not take into account the attribute values in the input and model graph. The similarity of two vertices or edges is considered by vertex-to-vertex mapping.

Definition 7: Given attributed graphs G_1 and G_2**,** and a graph isomorphism *f* from G_1 to G_2:

- *vertex mapping cost (vertex content similarity)* is value $vs(f) = \sum_{v \in V_1} VertexSim(v, f(v))$
- *edge mapping cost (edge content similarity)* is value $es(f) = \sum_{e \in E_1} EdgeSim(e, e')$, where $e = (v_1, v_2), e' = (f(v_1), f(v_2))$
- *total mapping cost (content similarity)* is $cs(f) = vs(f) + es(f)$

Now, we can define the concept of inexact subgraph matching by combining the subgraph isomorphism and edit operations definitions.

Definition 8: *Error-correcting subgraph isomorphism f* from attributed graph G_1 to attributed graph G_2 is a pair $f=(S, f_S)$ where

a) *S* is an edit sequence such that there exists a subgraph isomorphism from edited graph *S(G)* to G_2

b) f_S is a subgraph isomorphism from *S(G)* to G_2

Finally, we can define distance from a model graph to a similar subgraph in the input graph. We are interested in the error-correcting subgraph isomorphisms with

minimal total edit cost (expressing structural difference of the two graphs) and minimal total mapping cost (capturing similarity of mapped vertices and edges). However, different error-correcting subgraph isomorphisms match different subgraphs in the input graph. Thus, our results are set of matching subgraphs ordered by the distance from the model graph. The idea of the previous and next definition is sketched in Figure 7.

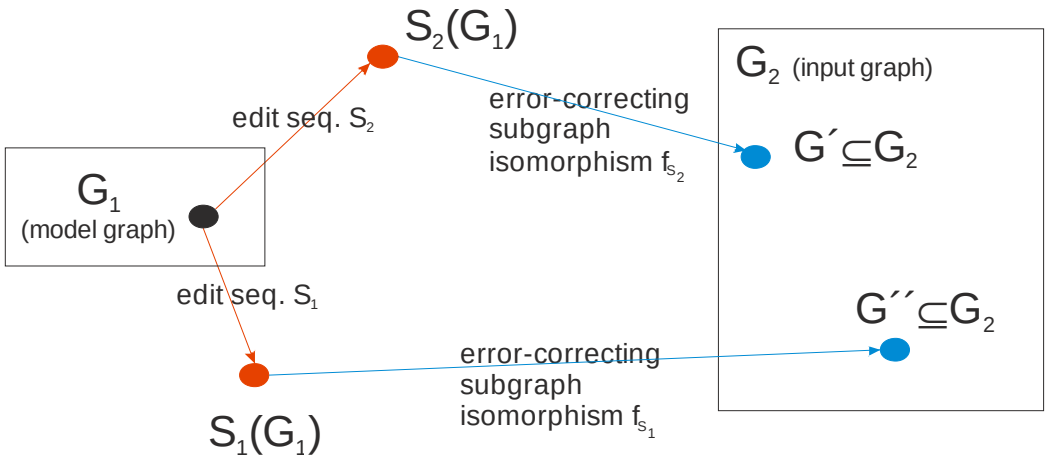

Figure 7: Error-correcting Subgraph Isomorphism

Definition 9: Given attributed graphs G_1 and G_2, and error-correcting subgraph isomorphisms $f_i = (S_i, f_{S_i})$ from G_1 to G_2, we define:

- *matching cost* for each error-correcting subgraph isomorphism $f = (S, f_S)$ as $C(f) = edc(S) + cs(f_S)$
- *matching cluster* for each error-correcting subgraph isomorphism $f = (S, f_S)$ as: $Match(G_1, G_2, f) = ((S(G_1))^{f_S}, C(f))$, i.e. a two-tuple of matching subgraph and the associated matching cost.
- *matching results* as an ordered set of matching results over all error-correcting subgraph isomorphisms f_i: $Output(G_1, G_2) = (X, \leq)$ where $X = \{Match(G_1, G_2, f_i) \mid f_i = (S, f_S)\}$ and $\leq$ is total ordering according to the matching cost associated with each matching cluster in X.

The first element in the *Output* set has minimal matching cost over all error-correcting subgraph isomorphisms from G_1 to G_2 and is the best match. The

associated $f = (S, f_S)$ is called *optimal* error-correcting subgraph isomorphism from G_1 to G_2 and the $C(f)$ can define distance from the graph G_1 to $\underline{G_2}$.

3.3.4 Search Algorithm

Given a model graph $G_M(V_M, E_M, \upsilon_M, \varepsilon_M)$, an input graph $G_I(V_I, E_I, \upsilon_I, \varepsilon_I)$, tables of similarities between vertices and edges of both graphs and costs for edit distance operations, an algorithm based on *A** search [59] is used for finding matching results between the model graph and similar subgraphs in the input graph as defined above. The algorithm explores all the possible mappings between vertices in V_M to vertices in V_I, starting with a single vertex-to-vertex mapping and iteratively extending the mapping by adding new vertex-to-vertex assignments until all the model vertices are mapped. In other words, a mapping is:

$$p = \{(v_1, w_1), (v_2, w_2), ..., (v_k, w_k) \mid v_i \in V_M \cup \{\$\}, v_i \neq v_j, w_l \in V_I \cup \{\$\}, w_l \neq w_m, (v_i, w_i) \neq (\$,\$)\}$$

A mapping is called complete if all vertices from V_M are mapped; otherwise it is referred as *partial*. Let us denote vertices included in the mapping as $V_p^M = \{v \mid (v,w) \in p\}$ for the model graph and as $V_p^I = \{w \mid (v,w) \in p\}$ for the input graph.

A complete mapping implies a set of edit operations *S* on the model graph and gives a vertex-to-vertex relation between the edited model graph $S(G_M)$ and an induced subgraph with vertices V_p^I in the input graph. In other words, the complete mapping *p* corresponds to an error-correcting subgraph isomorphism $f = (S, f_S)$. Analogically, cost of the complete mapping *p* corresponds to the matching cost *C(f)* of the error-correcting subgraph isomorphism *f* and will be denoted as *C'(p)* for mappings.

The cost of partial mapping can be computed iteratively. Assume we have a mapping $p = \{(v_1, w_1), (v_2, w_2), ..., (v_k, w_k)\}$ and we want to add a new vertex-to-vertex mapping (v, w), such that $v \in V_M \cup \{\$\}, \forall i : v \neq v_i$ and $w \in V_I \cup \{\$\}, \forall i : w \neq w_i$ and $(v, w) \neq (\$,\$)$, then the cost of extended partial mapping is:

$$C'(p \cup (v,w)) = C'(p) + vc'((v,w)) + \sum_{(v_i, w_i) \in p} ec'((v,w),(v_i,w_i))$$

where:

$$vc'((v,w)) = \begin{cases} VertexSim(v,w) & \text{if } v \neq w \neq \$ \\ c(vertexDelOp) & \text{if } w = \$ \\ c(vertexAddOp) & \text{if } v = \$ \end{cases}$$

$$ec'((v,w),(v_i,w_i)) = \begin{cases} EdgeSim((v_i,v),(w_i,w)) & \text{if } (v_i,v) \in E_M, (w_i,w) \in E_I \\ c(edgeDelOp) & \text{if } (v_i,v) \in E_M, (w_i,w) \notin E_I \\ c(edgeAddOp) & \text{if } (v_i,v) \notin E_M, (w_i,w) \in E_I \end{cases}$$

The function *vc'* return similarity between the vertices *v*, *w* or edit cost of deleting/adding a vertex if one of them is an empty vertex. The vertex *v* may be linked with some of already mapped vertices in the model graph G_M and, similarly, *w* may be connected to vertices in V_I^p. Function *ec'* compares the connectivity for each assignment in *p* and adds a penalty of deleting/adding an edge if the edge exist only in of the graphs; otherwise similarity of both edges is calculated.

An example of extending a partial assignment is shown in Figure 8: given a partial mapping *p={(m1,i1),(m2,i2)}*, the cost *C´(p)* will be increased by, for example:

- *VertexSim(m3,i5)* and *EdgeSim(e2,f4)* if mapping is *(m3,i5)*
- *VertexSim(m3,i6)* and edge delete cost for edge *e2* if mapping is *(m3,i6)*
- *VertexSim(m3,i4)* and *EdgeSim(e2,f1)* and edge add cost for edge *f2* if mapping is *(m3,i4)*

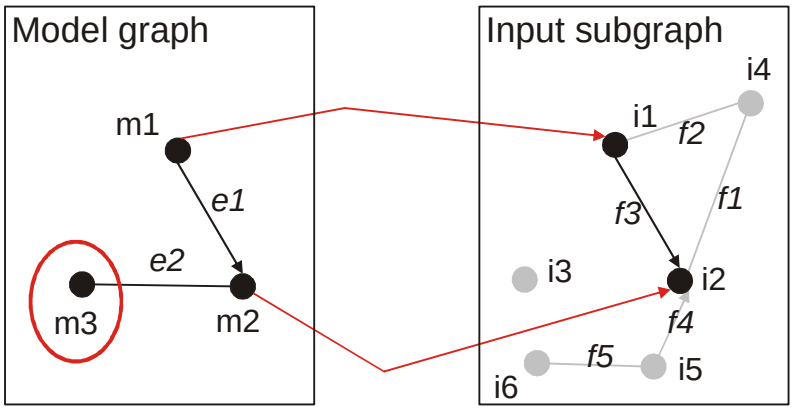

Figure 8: Extending partial mapping

The *A** algorithm starts with a vertex $v_1 \in V_M \cup \{\$\}$ from the model graph and maps it to all possible vertices in $w_i \in V_I \cup \{\$\}$. Thus, our state represents partial mappings $p = \{(v_1, w_i)\}$. In the next iteration, a mapping *p* with the lowest cost *C'(p)* is selected and the mapping is expanded by another assignment into

$p^2 = \{(v_1, w_i), (v_2, w_j)\}$ as described above. The process continues until a complete mapping p^k is found. This represents an optimal error-correcting subgraph isomorphism from G_M to G_I with minimal associated cost $C'(p^k)$. The cost is minimal, because the algorithm expands a partial mapping with the lowest cost in every step. The search continues exploring remaining partial mappings until another complete mapping p_2^l is found. Likewise, p_2^l is the mapping with lowest cost among remaining mappings and therefore second best matching cluster. Thus, following this process, the algorithm generates a set of all complete mappings ordered by their cost.

The implementation based on *A** is summarized in pseudocode in Figure 9. The cost of partial mapping *p* is extended by an estimate *h(p)* of cost of future mappings: $g(p) = C'(p) + h(p)$. The choice of heuristic function *h(p)* is discussed in section 3.3.5. All partial mappings are held in the *OPEN* queue and sorted according to their *g*-cost. If *p* is a complete mapping, then *p* is added to solutions and process continues with search for next complete mapping. The procedure ends when enough solutions have been found or the *OPEN* queue is empty. Note that the algorithm always finds a solution if the input graph is non-empty.

```
1. Generate initial mappings p, calculate g(p)
   and add them to OPEN

2. while OPEN not empty do

3.    select p with minimal g(p) from OPEN
      and remove p from OPEN

4.    if p is complete mapping then add p to solutions
      and goto (2)

5.    if (size of solutions > accept_solutions) then exit

6.    expand p into mappings p_i as described in the text,
      calculate g(p_i) and add them to OPEN

7. end
```

Figure 9: Graph Matching algorithm based on A*

Given a model graph with *n* vertices and an input graph with *m* vertices, the time complexity in the worst case without considering vertex add and delete operation is computed as follows: The worst case happens when every partial mapping gets extended. The first model vertex is mapped to *m* vertices in the input generating *O(m)* mappings. Each of these mappings will be extended by adding a new assignment

resulting into *O(m(m-1))* mappings. The process repeats *n* times until all the model vertices are mapped. For the *n*-th model vertex $O(m^n)$ mappings are generated. Hence, the sum of all mappings over *O(n)* model vertices is $O(nm^n)$. Furthermore, the algorithm executes *O(n)* times the *ec'* function for testing edge differences for each mapping. Consequently, the final time complexity is $O(n^2m^n)$ in the worst case. Allowing vertex delete operation adds one more state for each extension of partial mapping resulting into total time complexity $O(n^2(m+1)^n)$.

Any unmapped vertex from the input graph can extend the partial mapping if vertex add operation is perform. Enabling these vertex edit operations and assuming *k* inserts the time complexity results to $O((n+k)^2(2m+1)^{n+k})$ in the worst case.

3.3.5 Heuristics

Given a partial mapping *p*, heuristic function *h(p)* provides an estimate of the future cost of extending *p* into the complete mapping. The *h(p)* must be admissible (i.e. it should not overestimate the real cost) and monotonic [59], because we do not use the *CLOSE* list in the search algorithm. Although calculation of *h(p)* increases the time complexity, its effect can significantly decrease the number of expansions and thus decrease the exponential execution time.

A simple heuristic based on vertex and edge similarities can be calculated in *O(nm)* for each mapping. Given a partial mapping $p = \{(v_1, w_1), (v_2, w_2), ..., (v_k, w_k)\}$ the estimate of future cost is calculated as:

$$h(p) = \sum_{v \in V_M \setminus V_p^M} \min_{w \in V_I \setminus V_p^I} (VertexSim(v, w)) + \sum_{e \in E_M \,\&\, e \text{ unmapped}} \min_{f \in E_I \,\&\, f \text{ unmapped}} (EdgeSim(e, f))$$

Each unmapped vertex from the model graph is approximated by the smallest similarity with an unmapped vertex from the input graph. The same estimate can be computed for edges. Note that an edge is considered as unmapped if at least one of its vertices is not yet mapped. Such *h(p)* is admissible (it takes minimum over vertex and edge similarities for each model vertex and edge) and monotonic (satisfies triangular inequality). Similar heuristic function based on vertex attributes was successfully applied in [31].

The heuristic above does not take into account the edit costs that may be introduced by future mappings. [44] suggests a function *h(p)* that estimates future structural similarity. Each unmapped model vertex $v \in V_M \setminus V_p^M$ will be mapped to an

unmapped input vertex $w \in V_I \setminus V_p^I$ in the future. This may introduce an edge delete or add operation. Edge penalties can be approximated by considering the already mapped vertices by the partial assignment *p*: for each assignment $(v_i, w_i) \in p$, a model edge $(v, v_i) \in E_M$ must be mapped to an input edge $(w, w_i) \in E_I$, otherwise an edge delete or add cost is added. The estimate for model vertex *v* is the minimum of edge edit costs over input vertices *w*. Sum of the values for all unmapped model vertices is the final estimate. The heuristics can be calculated in *O(nm)* for each mapping.

Furthermore, [41] suggests an extension to the latter heuristics leading to more precise estimate of structural difference and yet better performance. Authors add an approximation of edit operations among unmapped vertices.

Both heuristics discussed in detail above are employed in our implementation. Work in [60] experimentally confirmed that the *A**-based search with this heuristic function performs reasonably well on graphs of moderate size (dozens of vertices) – it is worth noting that the graphs in our particular domain are certainly within this range (see Table 4).

3.3.6 Choice of Parameters

The DOCE matching algorithm is lead by content similarity (see section 3.3.2) and structural similarity (see section 3.3.3) while choosing the vertex-to-vertex assignments. The balance of these similarities, the choice of weights used in the content similarity calculation and the costs of edit operations significantly influences the matching results. Different weight sets stress different features and edit costs express how loosely or strictly is the graph structure preserved. For example, for a cluster, that strictly follows the discussion structure, higher weights for the shape and link type, the “Same user?” features and expensive edit cost may lead to better results. Whereas higher weights for the TagHelper features and the text length, and low costs of edit operations are more appropriate for a cluster description based purely on text content of contributions. Thus, the selection of parameters depends on the desired goals of the search. They may be selected by the user according to the cluster type or learned from annotated examples per cluster type as suggested in the section 3.5. However, the user choice of the parameters may not always be possible (e.g. a teacher

may not be interested in setting any parameters) and therefore we attempted to design a default set of parameters.

The weights for features were selected according to experience from the shape and pair-shape analysis [9], the unsupervised clustering experiments and qualitative analysis of the *1st HUJI* dataset. We evaluated several different feature sets as described in the experiments section.

The choice of costs for the edit operations is in general difficult question as there are no guidelines for selecting appropriate values [37,60]. Furthermore, the DOCE algorithm differs from other inexact matching algorithms by including the content similarity and thus the balance between content and structural similarity must be tackled.

We address this problem by a *heuristical estimate*. The matching cost is calculated as sum of the content similarity and the edit costs (see Definition 9). In order to balance the influence of the content and structural similarity, we estimate the edit cost based on the following idea. We would like to assure that the matching cost of a cluster *C* with *very high* content similarity and *small structural difference* (e.g. one or two edit operations) is about the same as the matching cost of a solution *C'* with *good* content similarity and *no edit operations*. In other words, even a cluster with slightly different structure from the model can be ranked as highly relevant if it has very good content similarity. The edit cost is calculated from the following equation:

$$CS_{\min} + \eta \cdot EC = CS_{\pi},$$

where G_1, G_2 is the model graph and input graph, respectively. The left side of the equation approximates the cost of the cluster with the best content similarity and several edit operations (cluster *C* above). CS_{min} is the minimal content similarity over model vertices and edges:

$$CS_{\min} = \sum_{v_1 \in G_1} \min_{v_2 \in G_2}(VertexSim(v_1, v_2)) + \sum_{e_1 \in G_1} \min_{e_2 \in G_2}(EdgeSim(e_1, e_2)) \ ,$$

the η is the expected number of edit operations and *EC* is the edit cost. The right side approximates the good cluster with no edit operations (cluster *C'* above). This is calculated as sum of π-th percentiles of content similarities between model/input vertices and edges for each model vertex and edge, respectively: $CS_{\pi} = \sum_{v_1 \in G_1} P_{\pi}^{v}(v_1) + \sum_{e_1 \in G_1} P_{\pi}^{e}(e_1)$ where $P_n^{\ v}(v_1)$ is π-th percentile for $VertexSim(v_1, v_2)$

values (over all input graph vertices v_2) and, analogically, $P_n^e(e_1)$ is π-th percentile for $EdgeSim(e_1, e_2)$ values (over all input graph edges e_2). The expected number of edit operations η is approximated by squared root of model vertices. The rational behind is that there is a lower chance of finding a structurally exact match for larger models.

The value π, specifying the percentile, is a parameter of the algorithm (referred to as *DOCE parameter* in the rest of the document) and balances the content and structural similarity with respect to the size of the model graph. Lower values prefer matching based on the content, whereas higher values lead to higher edit costs and thus stress the graph structure. For example, setting the value to π = 50 means that any cluster *C*, that has at least one edit operation, will have a matching cost greater than first 50% of structurally exactly matching clusters *C'* if they are ordered by their content similarity value (approximated as described above).

We do not distinguish cost of add and delete operations as there might be noise in both, model and input graphs. Thus, the edit cost for edge (node) add and delete operation is set as maximum of the *EC* value and maximum edge (node) similarity, respectively. This is in order to penalize structurally different matches, as the *EC* value may be less than the maximums for low π.

3.3.7 Limitations with Respect to Argunaut's Clusters

The algorithm matches subgraphs similar to the model example. Therefore, if there are clusters in the input map, that has exactly the same structure as the model example, the DOCE will prefer these clusters to clusters with different structures (e.g. different size).

For example, consider a model of cluster type *T* consisting of a chain of 3 vertices (e.g. *a-b-c*) and a discussion map that *contains* subgraphs with the *same* structure as the model. Furthermore, there are also occurrences of a cluster C_1 of type *T* that has 4 vertices *(a'-b'-c'-d')* and a cluster C_2 of type *T* with 2 vertices (*a'-b'*).

First, the cluster C_1 will be matched if a "vertex add" operation (and possibly "edge add" operation) is performed (assuming that the *a-b-c* graph was matched to *a'-b'-c'*). However, there is no cue telling the algorithm which vertex should be added. In other words, any vertex in the discussion could be added with the same cost (i.e.

the cost of the “vertex add” operation) and there is no information associated with the model in order to distinguish them. Therefore, the “vertex add” operation was *disabled* for the experiments as it leads to many false positives (every cluster (*a’-b’-c’*) plus any other shape in the discussion).

Second, in order to match the cluster C_2 vertex delete „and“ edge delete operations have to be performed (e.g. assume that we delete the node *c* from the model while matching the *a-b* to *a’-b’*). The match will rank well, if (1) the deleted vertex and edge are significantly dissimilar from neighboring vertices and edges of the vertices *a’* and *b’*, respectively, and (2) the edit cost is very low. However, we experimentally observed that such strict borderline between a cluster and the rest of the discussion does usually not occur in the discussions.

Thus, the algorithm matches clusters of the same size as the model graph if the discussion contains structurally same clusters. However, our cluster may vary in the size for the same cluster type, and the discussions typically contain clusters of the same structure as the model. We suggest an extension to tackle this property of the DOCE algorithm as discussed in section 3.5.

3.4 Implementation

We implemented the DOCE algorithm in Java by ourselves, including the inexact matching algorithm. To our knowledge, there are two inexact graph matching libraries available: (1) the *GUB* toolkit in C [61] – an implementation of the *ED* method written as part of research on extending the *ED* method to many-to-one matching [60]; and (2) an implementation of an iterative matching algorithm in Java [62] used for evaluation of work presented in [63]. We decided not to reuse the GUB toolkit because our application of the *ED* method requires many customizations such as incorporation of the content similarity. Furthermore, the MI and the discussion tools used on the project run on the Java platform.

In order to integrate the DOCE into the MI, a programming interface was designed in cooperation with the UDE – the project partner responsible for the MI development – and is sketched in Figure 10. DOCE implements this interface and uses the same data structures as the MI. In the experimental setup, the models and input graphs were stored in a MySQL database for its convenience and the algorithm was run from the command line.

```
/**
 * DOCE Matching Algorithm
 *
 * @author Jan Miksatko
 * @author Adam Lucarz
 */
public interface MatchSubgraphAlgorithm {
    /**
     * General method for DOCE searching for serveral patterns
     * over several discussion maps
     * (currently only one-to-one search is supported)
     *
     * @param inputs discussion maps to search on
     * @param models example clusters
     * @param additional parameters specified by user
     * in the UI dialog (Advanced Search)
     * @return ordered list by ranking of matches {@link MatchResult}
     */
    public List<IMatchResult> match(List<AbstractGraph> inputs,
        List<AbstractGraph> models, Properties params);
}


/**
 * A matching result produced by DOCE
 *
 * @author Jan Miksatko
 */
public interface IMatchResult extends Comparable<IMatchResult> {
    /**
     * Discussion Map ID.
     * @return ID of dicussion map containing the matched cluster.
     */
    String getInputMapId();

    /**
     * Model graph ID.
     * @return ID of pattern searched for.
     */
    String getModelMapId();

    /**
     * Ranking of the match
     * @return Ranking of this match.
     */
    double getRanking();

    /**
     * Vertex IDs representing a matching cluster.
     * @return Set of ID_KEYs of vertices representing the match.
     */
    Set<String> getMatchedCluster();
}
```

Figure 10: DOCE programming interface

The following steps were taken in order to ensure the correctness of the experimental evaluation:

- The experimental framework is based on standard, well-evaluated technologies, for example Java 5.0 or Apache libraries.
- Run-time assertions (about 100) were used across the code where eligible. Extensive logging of all operations and successive analysis of the logs was employed to ensure the correctness of the experiments.

- The structural matching (i.e. the ED method itself) was verified on test cases containing simple graph structures. Another set of tests was written for the similarity functions and the results were compared with an implementation in Matlab.
- The experimental results were manually analyzed and visualized in the Digalo maps to verify their correctness.
- In order to ensure the objectivity of the experiments, all annotated data were used as received from the pedagogical experts without leaving out any annotations, including the borderline ones for example. Furthermore, the experiment sets were designed so that the DOCE experiments are not biased to a particular model and a discussion topic.

3.5 Extensions

This section covers several suggestions on extending the DOCE algorithm with respect to the cluster detection in students' e-discussion.

Using a set of models against one discussion map and then merging the results of each search can improve the performance of the algorithm. First, this approach addresses the issue with matching smaller/larger clusters than the model graph (see the section 3.3.7). The set may contain models with various sizes and thus each DOCE run will detect clusters with size similar to the model size. The results are then merged according to the normalized ranking and thus matching clusters with different sizes are included. Second, different models may lead to different results, although the stability of the algorithm turned out to be quite high (see the Experiments section). The search accuracy can be improved by including into the final results set only matching clusters that (a) occur in several result sets, i.e. have enough support from different models (a voting mechanism), or (b) have high average ranking over several models.

Expert's knowledge can be associated with each cluster type by customizing the parameters of the algorithm and may lead the algorithm to better results. For example, for detecting cluster types, which focus on the discussion structure, high π and weights stressing discussion attributes are more appropriate. The parameters can be either specified by user or learned from enough annotated examples by parameter

optimization algorithms (e.g. [27]) with *Annotations covered* or *Search accuracy* measures as target function (see section 4.2).

Although the performance of the algorithm is not a limitation in the case of cluster detection in the e-discussions, the execution time can be further lowered by considering only limited set of vertices in the input graph while doing the vertex-to-vertex mapping. For example, only vertices from a neighborhood of predefined size or only contributions within predefined time interval with respect to the partially matched cluster in the input graph can be considered. Although such heuristics is destructive and turns the algorithm into an approximate solution, it could be successful for larger e-discussions where the debate occurs only in a part of the discussion.

Chapter 4

Experimental Evaluation

This section describes the evaluation that we performed to quantitatively verify whether the DOCE algorithm can succeed in detecting clusters defined by the pedagogical experts in real e-discussions.

We designed an evaluation methodology that uses a subset of the full set of maps. In particular, the specialists analyzed a set of 27 discussion maps, annotating cluster examples (referred to as "annotations" throughout the document), and then we used the cluster examples in the maps as input to the DOCE algorithm to evaluate how well the algorithm found the cluster examples in the other annotated maps of the subset. We also designed a set of experiments to compare performance among DOCE algorithms with different feature sets, to compare the approach with a random algorithm (as there is no other relevant matching algorithm) and that analyzes the performance with respect to cluster types and discussion maps.

We had hoped the experts would be able to provide us with a relatively large set of models (say dozens) per cluster type so that the DOCE algorithm could be run against hundreds – or even thousands – of other discussion maps, all of which had also been annotated (and evaluated for interrater reliability) with the clusters of interest. Then, the results from the algorithm could have been quantitatively compared against the original annotations of the experts. In addition, it would have been useful to ask the experts to analyze the DOCE results that did not agree with their assessments (again applying interrater reliability), in cases in which the DOCE algorithm discovered clusters the experts had not identified. Such an evaluation methodology is the most exhaustive and objective imaginable, but such an approach would also require an extensive amount of time and effort from the experts – time and effort that simply could not be obtained. However, we believe the more limited approach we have designed and executed is a realistic and reasonable approximation of such an extensive and objective analysis.

4.1 Datasets

The HUJI pedagogical experts annotated two sets of maps from the real discussions – the 1st and 2nd HUJI datasets (see section 2.1) – 12 maps in the first set and 15 maps in the second set for the 3 most important clusters from HUJI's point of view: *Clarification of opinion following feedback* - *CoOp* (4+14 annotations), *Chain of opposition* - *CoO* (7+13 annotations), *Argument* + *evaluation* - AE (8+28 annotations).

The annotations are connected graphs with 2 to 5 vertices and thus the DOCE was evaluated only with respect to the connected clusters. However, the evaluation is reasonable as (1) the connectivity is a property of these cluster types; (2) only good, non-erroneous (i.e., fully connected) examples will typically be used as a model in the MI. Note that the annotations of the same type have *mostly* the same size and therefore the DOCE evaluation is not influenced by the limitation discussed in the section 3.3.7. Nevertheless, the examples that differ in the size from the remaining annotations for the same cluster type were *kept* in the dataset, although they decreased the performance of the algorithm.

We used all available annotated maps and designed an experiment set per each cluster type and map as follows. Given a cluster type *T* and a discussion map *A*, all annotations of cluster type *T* from all maps except for the map *A* itself were used as models and run against the map *A*, i.e. an *experiment set (T, A)* is a set of *n* DOCE searches $\{M_1, M_2, \ldots, M_n\} \times A$, where M_i are annotations of cluster type *T* except for the annotations in the map *A*. This approach avoids bias to a particular model and tests the "generalization" of the algorithm – the method should perform well on any model representing a cluster of type *T*. Furthermore, topics in the discussions vary and thus the evaluation is not biased to language in one domain as the source discussion for a model M_i and the map *A* may debate different topics.

Only maps having two or more annotations were used as input maps (map *A* in the formula). The experiment sets are summarized in the Table 9. The sets are split according to the dataset because each of them was annotated by a different annotator and thus only models from one dataset are used against the input maps in the same dataset. The "Models" column represents the number of models (M_i) used against the input map (*A*). Notice the low number of annotations per input map.

Cluster Type	Dataset	Input Map	Models	Annotations in Input Map
Clarification of opinion following feedback	1st HUJI	biology_experiments3	2	2
	2nd HUJI	0grup1	13	2
		0internet5	11	4
		Butterfly1	13	2
		Butterfly2	13	2
Chain of opposition	1st HUJI	genetics_cf11	4	3
		genetics_cf3	5	2
	2nd HUJI	0arnavon	5	8
		0chin	10	3
Argument + evaluation	1st HUJI	biology_experiments4	6	2
		biology_experiments5	6	2
		biology_experiments7	6	2
	2nd HUJI	0arnavon	22	6
		0chin	24	4
		0chinchila	23	5
		0internet2	26	3
		Butterfly1	23	5
		Butterfly2	26	2

Table 9: Experiment sets

4.1.1 Additional Information about the Datasets

Currently, a single expert annotated the 1st HUJI set and another single expert annotated the 2nd HUJI set. An experiment was done on the 1st dataset in which we asked another expert from HUJI and an expert from Exeter to independently annotate the maps. Unfortunately, the agreement was very low on both intra- and inter-institutional levels and thus high interrater reliability has not yet been achieved. Therefore, we contacted other projects working with argumentation maps in order to see if we could find another well-annotated data source, one with interrater reliability that could be used as a "gold standard" dataset. Unfortunately, we did not find such a project.

Despite this shortcoming, we argue that the dataset and annotations are sound and the evaluation meaningful because the DOCE detects clusters that are similar to the provided model – hence the algorithm adapts to the style of the annotator. It seems likely, also, that obtaining a high level of interrater reliability for such an arduous and inexact task as identifying "meaningful" clusters will be very hard to attain indeed. Note, for instance, that many annotations were marked as *borderline* examples, especially for the *AE* cluster type. Despite the fact that such weak annotations could *negatively* influence the results of our experiments, we decided to keep and use *all* of the annotations received.

4.2 Evaluation Methodology

The DOCE approach functions similar to web search engines and therefore the evaluation design is motivated by the Information Retrieval (IR) literature [40,64]. Typically, the first 10 results (Top10) are considered in the web search evaluation. The results are compared with user feedback, for instance, a group of experts rates the results in the Top10 as *highly relevant*, *relevant* or *not relevant*. Several criteria are then calculated, for example, *Precision* measure as the number of highly relevant and relevant results divided by 10.

Likewise, we use the Top10 matching clusters (i.e. results) to evaluate the algorithm. In our case, the feedback is provided by the pedagogical specialist in the form of annotated clusters. The relevancy criterion is defined as follows:

Definition 10: A matching cluster C of the DOCE search on the input map M is called *relevant* iff there exists an annotation A in the input map M such that: $|A \cap C| \geq 70\% \cdot |A|$ and if $|A| \leq |C|$ then also $|A \cap C| \geq 70\% \cdot |C| \vee A \subseteq C$

A relevant matching cluster is called *perfect* iff it precisely matches an annotation in the map M.

An annotation A in the input map M is called *covered* iff the DOCE search results contain a matching cluster that is relevant to this annotation A.

In other words, a matching cluster is relevant if the overlap of vertices between the matching cluster and an annotation is 70 % (rounded) of the annotation size (e.g. if an annotation is {1,2,3}, then the cluster {2,3,4} is relevant, since vertices 2 and 3 appear in both the annotation and cluster, leading to 66% overlap, rounded to 70%). Furthermore, in order to filter too large matching clusters, if the annotation is smaller than the clusters, then the size of the overlap must be 70% of the size of the matching cluster or the annotation must be a subset of the matching cluster (e.g. if an annotation is {1,2,3}, then a cluster {2,3,4,5} is *not* relevant, although the overlap is within 70% of annotation size; however, a cluster {1,2,3,4,5} is relevant).

Due to the imprecision in the annotation process, the DOCE search cannot always match the exact clusters. We consulted with the pedagogical experts about the definition of the relevance rule, and it was fully accepted as satisfactory. The rule is

based on the following rationale – the goal of the cluster detection is to point the moderator to the portion of the discussion where an interaction of interest occurs. Therefore, even a non-exact match can satisfy this goal if a substantial part of the matching cluster overlaps with the desired (interesting) cluster. We calculate the "70% of size" as specified in Table 10 by rounding the value except for clusters of size two where a *stricter* rounding up is used in order to avoid matching just one shape (i.e. annotations of size two must be matched *perfectly*).

Cluster size	70% Unrounded	70% Rounded	"70% Strict"
1	0.7	1	1
2	1.4	1	2
3	2.1	2	2
4	2.8	3	3
5	3.5	4	4

Table 10: The 70% relevancy rule

We use several metrics for the quantitative evaluation of the results. Per each DOCE run, i.e. each model graph and input map, the following measures are computed:

- *Annotations Covered* represents the number of annotations that were covered by matching clusters in the Top10 divided by the count of annotations in the searched map. The value represents how many annotations were detected by the algorithm. This definition is equivalent to the *Recall* measure in the IR literature if all annotations are considered as all relevant documents [65].

- *Search Relevancy*, referred to as *Precision* in the IR domain, is the number of relevant matches in the Top10 divided by 10. It. expresses the portion of results that are relevant. Note that this value can be generally low because the input maps contain just 3.3 annotations on average.

- *Ranking Quality* measure, also known as *Average Precision* in the IR literature, is calculated as:

  $$AveP = \frac{\sum_{r=1}^{10} P(r) \cdot rel(r)}{|\, relevant\_matches \,|}$$, where r is rank, *rel()* is a binary function on the relevance of given rank (i.e. expressing relevant or not relevant) and *P()* is precision at a given cut-off rank. The function emphasizes returning more relevant documents earlier and therefore measures the quality of the ordering

of the results. For example, the relevant matches on positions (1,2,3) lead to calculation $AveP = \frac{\frac{1}{1}+\frac{2}{2}+\frac{3}{3}}{3} = 1$; whereas ranking relevant matches at positions (1,5,10) results into $AveP = \frac{\frac{1}{1}+\frac{2}{5}+\frac{3}{10}}{3} = 0.5\overline{6}$. In other words, the higher value means better ordering of the matching clusters, with the best value of 1.0 (all matches are on the top). A more precise measure, differing perfect and relevant matches, can be defined similarly to work in [66].

- *Stability* is used to compare two DOCE searches each with a different model of the same cluster type against the same map. The measure is calculated as the size of intersection of the two result sets. A matching cluster from the first result set is included in the intersection set if there is 70% overlap with a matching cluster from the second result set. High values mean that the DOCE algorithm produces very similar result sets independently from the selected model.

- *Perfect matches* represents portion of relevant matches that are perfect, i.e. the number of perfect matches divided by the number relevant matches in the Top10.

The *Annotations Covered* (i.e. *Recall*) is the key criterion as it is important to point the teacher/moderator to an interesting cluster in the discussion. We believe that the number of relevant matches (i.e. *Precision*) have lower significance since the humans are clever enough to filter the irrelevant matches, however, ideally, the algorithm should not miss a cluster in the discussion.

Each experiments set $\{M_1, M_2, \ldots, M_n\} \times A$ defined in Table 9 results into *n* DOCE searches and the measures described above are calculated for each search. The performance of the experiment set is calculated as an average over all the *n* searches. Furthermore, the stability measure of the experiment set is calculated as an average over all intersections between two results sets $(M_i \times A, M_j \times A)$. The overall performance of the algorithm is computed as an average over the experiment sets.

Unfortunately, there is no existing algorithm to that the DOCE method could be compared as this work is unique. Therefore, we designed a random matcher that generates a set of random cluster of the same size as the model size. This is a

reasonable constraint, because the DOCE prefers clusters of the same size as the model (see section 3.3.7) and the most of the annotations of the same cluster type are of the same size. The random generator was executed 100 times for each model/input pair *(M_i x A)* and the results were averaged.

4.3 Example of Algorithm Run

To provide context for the experimental evaluation, Figure 11 shows an example of a DOCE search. The *biology_experiments3* map in Appendix A is used as the input graph and an annotation of the *CoOp* cluster in another discussion is used as the model graph. The results of the algorithm are compared to the annotations by the HUJI experts in the input map (see Table 3).

```
Model example:
CoOp1 (from map biology_experiments2, vertices {3,9,10}, edges:
{(3,9), (9,10)})
Input map: biology_experiments3
Configuration: feature set Text / π=50.0
Cluster type: Clarification of opinion following feedback
Annotations in the input map: {{31,33,22}, {51,44,32}}

Results:
1. Cluster: [32, 44, 51] / Cost: 1.314
[vs=0.839/es=0.475/p=0.0/op=]
2. Cluster: [32, 38, 50] / Cost: 1.502
[vs=0.789/es=0.712/edc=0.0/op=]
3. Cluster: [32, 38, 59] / Cost: 1.521
[vs=0.959/es=0.563/edc=0.0/op=]
4. Cluster: [38, 50, 59] / Cost: 1.611
[vs=1.136/es=0.475/edc=0.0/op=]
5. Cluster: [22, 31, 33] / Cost: 1.682
[vs=1.295/es=0.387/edc=0.0/op=]
6. Cluster: [25, 32, 38] / Cost: 1.700
[vs=1.138/es=0.566/edc=0.0/op=]
7. Cluster: [1, 8, 15] / Cost: 1.773
[vs=0.948/es=0.825/edc=0.0/op=]
8. Cluster: [8, 15, 32] / Cost: 1.831
[vs=0.793/es=0.0/edc=1.039/op=ED;]
9. Cluster: [8, 15, 33] / Cost: 1.85
[vs=0.811/es=0.0/edc=1.04/op=ED;]
10. Cluster: [25, 32, 44] / Cost: 1.857
[vs=1.295/es=0.5625/edc=0.0/op=]

Summary:
Perfect Match: 1. [51, 32, 44]
Perfect Match: 5. [33, 22, 31]
Relevant Match: 10. [32, 25, 44]

Search Relevance: 30.0%
Average Precision: 0,567
Annotations Covered: 100%
```

Figure 11: DOCE search for *CoOp* cluster on map *biology_experiments3*

The search uses a feature set that stresses text attributes (see section 4.3), the DOCE parameter π is set to 50 and the resulting edit operation cost is *1.04* (see section 3.3.6).

The data associated with each result show the set of vertices matched (the numbers correspond to the IDs in the Digalo screenshot) and the matching cost. Furthermore, below each result, the details of the cost calculation are shown – sum of vertex similarities, sum of edge similarities, sum of edit costs (i.e. penalties) and edit operations performed on the model graph (*ED* stands for edge delete). The results are ordered by their matching cost.

The eighth and ninth match represent a subgraph that has an unconnected vertex and therefore an edge delete operation on the model graph was performed. In fact, the content similarity of both clusters is higher than the content similarity for the rest of the structurally exact matches; however, their structure differs by one edge from the model graph.

The input map contains two occurrences of the *CoOp* cluster and they are matched precisely by the algorithm as the first and fifth result. Therefore, the *Annotations Covered* measure is 100%. Furthermore, the last cluster is marked as *relevant* because it overlaps in 70% with the second annotation. Thus, the *Search Relevance* results to the value of 30%.

4.4 Experiments with Parameters

We designed several feature sets (FS) in order to evaluate the influence of different features in the content similarity calculation (see section 3.3.2) as shown in Table 11. We leveraged the experience from the shape/pair-shape analysis [9] and the unsupervised clustering experiments when designing these sets and choosing weights. The values in the table represent weight of a particular feature in the content similarity calculation for vertices and edges, respectively.

Feature Set	Vertex Features				Edge Features			
	Shape Type	Text Length	Tag Helper	Annota-tions	Link Type	Link Dir	Same User?	Annota-tions
Baseline	50%	50%	0%	0%	33%	33%	33%	0%
(+annotations +TagHelper)	25%	25%	25%	25%	25%	25%	25%	25%
Observations	10%	30%	30%	30%	15%	15%	35%	35%
Text	10%	20%	50%	20%	15%	15%	35%	35%

Table 11: Feature sets for the DOCE experiments

The feature sets have following meaning:

- The *Baseline* FS uses only attributes available from the discussion map.
- The second FS adds annotations from the shape/pair-shape level analysis [9] and attributes obtained from the TagHelper text analysis [16]. The weights are distributed equally. This FS evaluates the effect of including the annotations and the TagHelper features.
- The "*Observations*" FS includes all the features from the second FS; however, the weights are based on observations from the qualitative analysis of the 1st HUJI dataset. We compared annotations of the same cluster type among each other and observed that the clusters share similar (1) annotations on vertices and edges, and (2) user exchange patterns. On the other hand, they exhibit low similarity in the shape and link types, and the link direction. This is probably caused due to students' mistakes.
- Finally, the last FS strongly prefers *text* attributes (i.e. TagHelper features) while having the same weights as the third FS for the edge features.

Note that the feature sets does not include the *time delta* edge feature. We experimentally concluded that this attribute slightly decreases the *Annotations Covered* and *Precision* measures for all the designed features sets. This confirms the experience of the pedagogical experts that contributions are often modified after their creation and therefore comparing timestamps of two contributions includes misleading information. Similarly, we experimented with removing *unigrams* and *bigrams* from TagHelper calculation as we suspected biasing the algorithm towards a

topic of the discussion (a model graph can be chosen from a discussion with a different topic than the input map). This hypothesis was not experimentally confirmed and, in contrary, the inclusion of mentioned TagHelper features leads to small improvement in the performance across all the measures for all feature sets.

For each FS, we experimented with different π={1,25,50,75,100} value although the influence of the π parameter can be evaluated only limitedly since the models are connected graphs.

The experiments evaluating the algorithm with respect to the different feature sets and different π values are summarized in Table 12. The measures are averaged over all experiment set results. Following conclusions can be drawn:

- The results improve from the *Baseline* FS towards the *Text* FS for each measure (except for *Ranking Quality*) and each π value. In other words, the inclusion of TagHelper features and shape/pair-shape level annotations along with manual choice of weights according to the qualitative analysis of the 1st HUJI set leads to the best performance. The difference is especially significant for lower π values (up to 14% for *Annotations Covered* measure). This is because low π prefers content similarity to the graph structure and hence a good choice of features is important. Furthermore, preference of TagHelper features in the similarity calculation slightly improves the results (by 1-2% on average for *Annotations Covered* criterion). Thus, the *Text* FS is the best combination of features and weights.

- The results improve with growing value of π for all measures and all FS. This is not surprising since the annotations were connected graphs and therefore higher edit costs preserves the graph structure better.

- An interesting conclusion is that both, the DOCE parameter and the feature set choice, does not influence the quality of ranking.

Feature Set	π	Annotations Covered (Recall)	Retrieval Accuracy (Precision)	Ranking Quality (Average Precision)	Stability (Average Intersection)
Baseline	1	**54.7%**	27.8%	0.52	3.0
	25	**56.5%**	28.2%	0.52	3.2
	50	**62.7%**	28.6%	0.51	4.3
	75	**71.7%**	32.4%	0.52	5.3
	100	**72.7%**	32.9%	0.52	5.5
(+ annotations + TagHelper)	1	**60.7%**	31.5%	0.56	3.5
	25	**61.4%**	31.4%	0.55	3.5
	50	**68.2%**	31.3%	0.56	4.6
	75	**74.8%**	34.6%	0.56	5.5
	100	**76.2%**	34.9%	0.55	5.7
Observations	1	**65.7%**	33.4%	0.58	4.3
	25	**65.7%**	33.3%	0.58	4.3
	50	**71.4%**	34.4%	0.56	5.1
	75	**75.4%**	36.2%	0.56	5.9
	100	**77.2%**	36.6%	0.56	6.1
Text	1	**69.0%**	36.2%	0.57	4.8
	25	**69.0%**	35.8%	0.56	4.8
	50	**73.0%**	35.8%	0.57	5.3
	75	**77.8%**	37.0%	0.57	6.0
	100	**79.0%**	37.3%	0.57	6.2

Table 12: Comparison of different configurations

Finally, Figure 12 lists values of minor parameters that are used in the experiments. The values were obtained based on data analysis and experimental evaluation.

```
// Size of result set
MAX_SOLUTIONS = 10;

// Allowed node edit operations
NODE_ADD_ALLOWED = false;
NODE_DEL_ALLOWED = true;

// Number of maximum node edit operations
MAX_NODE_ADD_DEL_OPERATIONS = 3;

// Normalization and outliner detection constant for text (MTL constant)
MAX_TEXT_LENGTH = 250;
// Normalization and outliner detection constant for time delta (the MTD constant)
MAX_TIME_DIF = 250 * 1000;
// Normalization constant used in Jaccard measure calculation for TagHelper features
TAGHELPER_FEATURES_NORM = 7;
```

Figure 12: Additional parameters used in the experiments

4.5 Results

This section presents overall evaluation of the DOCE algorithm. The results are compared to the random matcher and summarized with respect to the cluster types, datasets and discussion maps.

Configuration	Annotations Covered (Recall)	Retrieval Accuracy (Precision)	Ranking Quality (Average Precision)	Stability (Average Intersection)
Random Matcher	**21.3%**	6.6%	0.32	0.5
DOCE (Baseline, π=50)	**62.7%**	28.6%	0.51	4.3
DOCE (Text, π=50)	**73.0%**	35.8%	0.57	5.3
DOCE (Text, π=100)	**79.0%**	37.3%	0.57	6.2

Table 13: Comparison of DOCE algorithm to the random matcher

Table 13 compares the DOCE algorithm to the random matcher. The DOCE algorithm performs significantly better across all the measures and all configurations as confirmed by t-test (the change was very significant at $p<0.001$ level). The DOCE can match up to almost *60%* annotations more than the random method (i.e. *Recall*). The random matcher has also significantly lower *Precision* and *Stability*.

Furthermore, from the overall results following conclusions can be drawn:

- The *Precision* measure can be intuitively interpreted as that every third matching cluster is relevant. This value is low, however, it is worthy to note that the input maps contain 3.3 annotations on average (see Table 9).
- The stability of the DOCE algorithm with respect to different models is relatively high. On average 6 clusters (for the best configuration) are in common when comparing two results sets produced by two different models against the same map, despite the models can be from discussions with different topics.
- Detailed analysis of the results showed that, on average, more than 60% of relevant clusters are perfect matches (in comparison to 11% for the random matcher).

- A ranking having the same Average Precision as the last two DOCE configurations is (2,3,6), for example.

The results in the following tables, summarizing the evaluation of the algorithm per cluster type, dataset and input map, are based on the "default" configuration – the *Text* feature set and π*=50*. We decided to use the "neutral" value of π in order to stay unbiased from the parameter choice. The detailed results of all experiment sets for the configuration [Text; π=50] are included in Appendix C.

First, Table 14 summarizes the overall results per cluster type. The *Recall* for both, the *CoOp* and *CoO* cluster types, is relatively high (over 70%). Moreover, on average, every second cluster is relevant. The lower value of *Annotations Covered* measure for the *AE* cluster might be caused by many borderline examples among the annotations in contrast to the other cluster types. Furthermore, the drop in the *Precision* of the *AE* cluster type is because all the annotations are of size two and therefore each match must be *perfect* (according to the strict 70% rule – see Table 10). Thus, also the *Recall* value for the *AE* cluster type is based on the perfect matches.

Cluster Type	Annotations Covered (Recall)	Retrieval Accuracy (Precision)	Ranking Quality (Average Precision)	Stability (Average Intersection)
Clarification of opinion following feedback	**87.4%**	47.9%	0.67	5.2
Chain of opposition	**74.6%**	56.1%	0.67	4.0
Argument + evaluation	**64.4%**	20.1%	0.47	5.9

Table 14: Results summarized per cluster type for [Text FS; π=50]

Another view of the overall results contains Table 15 that summarizes the performance per dataset. It is interesting to observe that the maps with low value of *Annotations Covered* criterion, in particular *0arnavon, 0internet2, Butterfly1* are marked by the pedagogical experts as very messy and uneasy to annotated; for example the annotator commented the map *0arnavon* as:

> *"the map is a total mess, had to be extensively re-arranged so that relationships could be seen. Even so, it was still hard to read…"*

In other words, there is a chance that the annotator did a mistake and missed some clusters.

Map	Annotations Covered (Recall)	Retrieval Accuracy (Precision)	Ranking Quality (Average Precision)	Stability (Average Intersection)
0arnavon	**30.1%**	17.0%	0.29	2.7
0grup1	**95.5%**	45.5%	0.70	4.9
0chin	**68.3%**	45.0%	0.58	3.7
0chinchila	**72.7%**	36.4%	0.71	5.3
0internet2	**50.7%**	15.2%	0.44	5.2
0internet5	**82.5%**	68.0%	0.86	3.6
biology_experiments3	**100.0%**	35.0%	0.49	7.0
biology_experiments4	**91.7%**	18.3%	0.49	9.1
biology_experiments5	**100.0%**	20.0%	0.69	6.3
biology_experiments7	**33.3%**	6.7%	0.21	3.5
Butterfly1	**55.9%**	28.6%	0.52	4.3
Butterfly2	**99.0%**	39.8%	0.60	8.7
genetics_cf11	**91.7%**	72.5%	0.87	6.3
genetics_cf3	**90.0%**	66.0%	0.78	5.2

Table 15: Results summarized per map for [Text FS; π=50]

Finally, Table 16 presents the overall results per dataset. The difference in the average *Recall* might be caused by the fact that the 2nd HUJI dataset contains many annotations marked as borderline by the HUJI experts in contrary to the first dataset.

Dataset	Annotations Covered (Recall)	Retrieval Accuracy (Precision)	Ranking Quality (Average Precision)	Stability (Average Intersection)
1st HUJI set	**84.4%**	36.4%	0.59	6.2
2nd HUJI set	**67.3%**	35.5%	0.56	4.8

Table 16: Results summarized per dataset for [Text FS; π=50]

The figures Figure 13 - Figure 15 summarizes the results above for the most important criterion, *Annotations Covered*, in a graphical form along with results for random matcher and [Text; π=100] configuration that can be understood as the DOCE algorithm with manually tuned parameters. With respect to the optimized configuration [Text; π=100], notice the significant increase in the *Recall* for the map *biology_experiments7*.

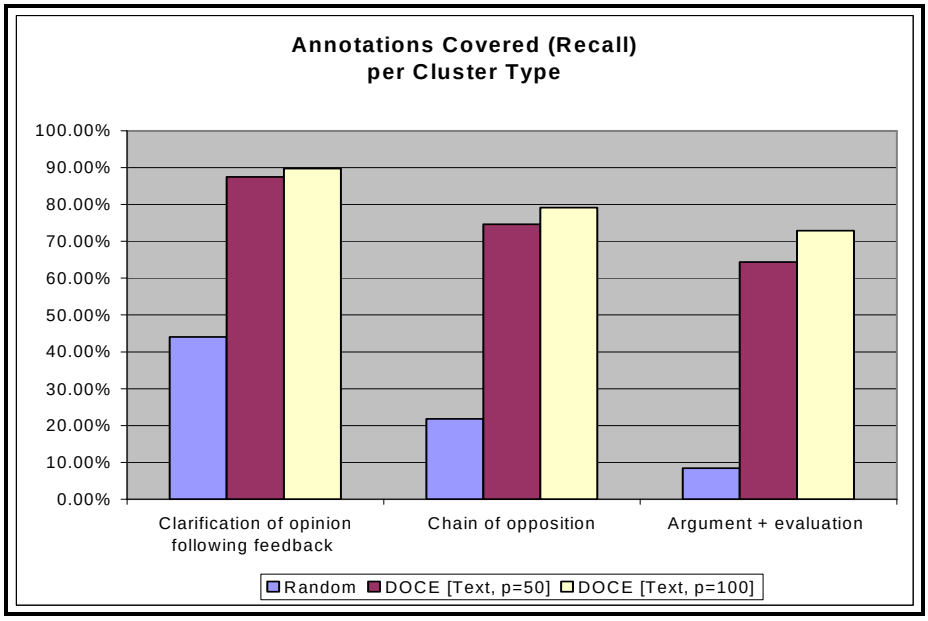

Figure 13: Recall per Cluster Type

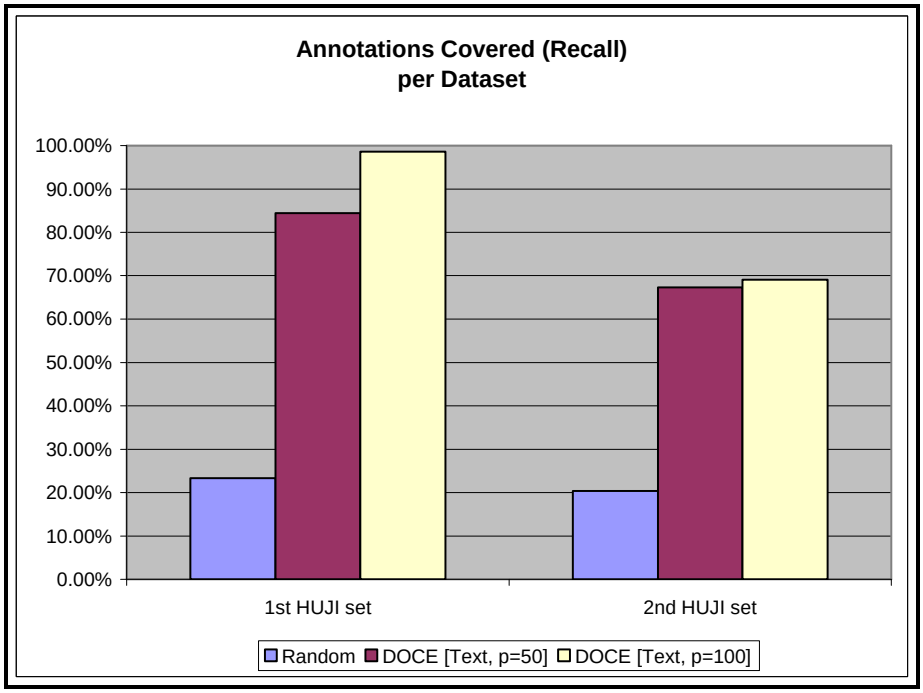

Figure 14: Recall per Dataset

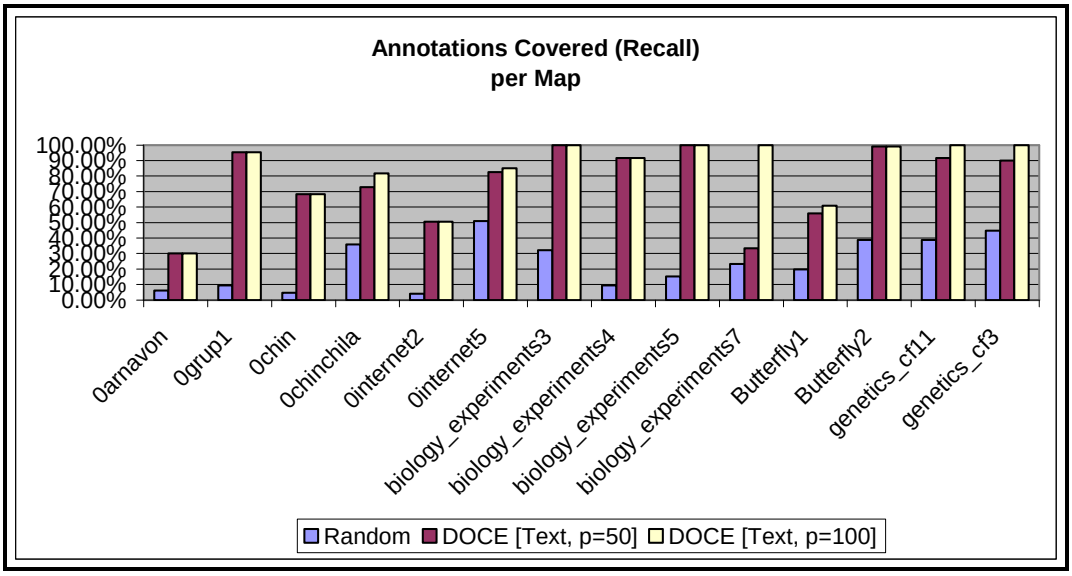

Figure 15: Recall per Map

4.6 Discussion

The algorithm demonstrated high *Recall* for all three cluster types, where two of them, the *Clarification of opinion following feedback* and *Chain of opposition*, covered over 70% of annotations. The experiments also showed relatively high *Stability* when searching with different models against the same map, i.e. the DOCE approach demonstrates good independence from the model graph that is used as the query.

The algorithm was compared to another approach, although a simple one, as there was no other algorithm available, and significantly over performed the random matcher.

Furthermore, the experiments with different features sets demonstrated that the inclusion of attributes from the TagHelper linguistic analysis and the shape and pair-shape level classifiers leads to improvement in the *Recall* and *Precision.* On the other hand, we were not able to evaluate the effect of the π parameter because of the connected models in the dataset. Similarly, no conclusions can be drawn about the behavior of the algorithm on unconnected models and cluster type that significantly vary from one another. Thus, an evaluation on another dataset with such properties is required.

Besides, the algorithm was evaluated on relatively small dataset and thus an evaluation on a larger set of maps is needed, in other words, more maps must be

annotated. However, the DOCE algorithm can be used as a tool for obtaining such annotations and thus significantly simplify this task for the pedagogical experts.

Chapter 5

Further Work

We are planning further evaluation of the algorithm on a larger dataset and for more cluster types. Currently, the annotations of the *4th HUJI set* for the three most important cluster types (*CoOp*, *AE* and *CoO*) are being prepared and the Exeter researchers are finishing their annotations on the *3rd HUJI set* for 9 types of clusters. The latter dataset provides a challenge for the DOCE algorithm because it contains unconnected clusters and clusters of the same type that vary in the size.

Furthermore, we are working on the integration of the algorithm into the Moderator's Interface in order to provide the pedagogical researchers with a tool for obtaining more annotations. Later, after qualitative evaluation of the results by the pedagogical experts, a new Awareness Indicators for the most successful cluster types may be added to the MI.

Our long-term goal is obtaining enough annotations in order to better understand the cluster types and pursue a "smarter" method, which can leverage more domain knowledge from the annotations and/or cluster understanding, such as supervised learning. For example, an ML classifier may be used for filtering results produced by the DOCE algorithm. Or, a supervised clustering could be performed – a method based on the same principle as unsupervised clustering – however, the distance measure is not specified *a priori* by a researcher, but learned from the annotated examples [67].

Chapter 6

Conclusion

Students in some classrooms are starting to use visual argumentation tools for e-discussions. In order to effectively moderate several simultaneous discussions, a tool providing feedback to the teacher is required. The Argunaut system help the moderator monitor the progress of the debate by Awareness Indicators that display interesting events in the discussion.

In previous work, machine-learning classifiers were designed for categorizing contributions and pairs of contributions in order to recognize student actions, such as using critical reasoning and raising and answering questions. We focused on an extended problem – analysis of larger segments of discussion maps representing interaction patterns that are of pedagogical interest. Detection of such clusters of contributions is a complex task because the data combine graph and text structure, the cluster types are imprecisely defined and we have limited source of annotations.

We explored several possible approaches and proceeded iteratively in the research from the unsupervised techniques, requiring minimum annotations towards supervised methods demanding dozens of annotations. The first method was pursued in order to provide the pedagogical experts with a tool for getting cluster candidates and thus ease the annotation process; however, the unsupervised clustering algorithm results were not promising enough.

In the next iteration, we designed the DOCE algorithm that uses an example of a cluster for finding similar clusters in the discussion maps. The method is based on an extension of edit distance inexact graph matching algorithm. The DOCE approach is looking for subgraphs in the discussion map that have the highest content similarity and lowest structural difference from the model. The content similarity calculation includes the discussion attributes, the text analysis performed by the TagHelper tool, and the classifications from shape/pair-shape level analysis.

We evaluated the algorithm on 27 real discussion maps for the three most important clusters that were annotated by the pedagogical experts. The DOCE algorithm was able to detect over 70% of annotated clusters. We used all models

available including the ones from discussions with different topics and the ones that were annotated as borderline examples. Furthermore, we compared the results with a random matcher, as there was no other algorithm available, and the DOCE significantly over performed this approach.

In sum, the experiments, although preliminary and on limited dataset, have shown promising results. However, deeper investigation and larger evaluation is required. The research on cluster analysis presented in this work is still initial. We intend to analyze the algorithm on a larger dataset and with more complicated clusters. Furthermore, the DOCE algorithm will be integrated into the Moderator's Interface and thus it can help pedagogical experts obtaining more annotations that may be later used for supervised learning, for instance.