Example-based feedback provision using structured solution spaces

Sebastian Gross*

Department of Informatics,
Humboldt-Universität zu Berlin,
Unter den Linden 6,
10099 Berlin, Germany
E-mail: sebastian.gross@hu-berlin.de
*Corresponding author

Bassam Mokbel, Benjamin Paassen and Barbara Hammer

CITEC Cognitive Interaction Technology Center of Excellence,
Bielefeld University,
Inspiration 1 (Zehlendorfer Damm 199),
33594 Bielefeld, Germany
E-mail: bmokbel@techfak.uni-bielefeld.de
E-mail: bpaassen@techfak.uni-bielefeld.de
E-mail: bhammer@techfak.uni-bielefeld.de

Niels Pinkwart

Department of Informatics,
Humboldt-Universität zu Berlin,
Unter den Linden 6,
10099 Berlin, Germany
E-mail: niels.pinkwart@hu-berlin.de

Abstract: Intelligent tutoring systems (ITSs) typically rely on a formalised model of the underlying domain knowledge in order to provide feedback to learners adaptively to their needs. This approach implies two general drawbacks: the formalisation of a domain-specific model usually requires a huge effort, and in some domains it is not possible at all. In this paper, we propose feedback provision strategies in absence of a formalised domain model, motivated by example-based learning approaches. We demonstrate the feasibility and effectiveness of these strategies in several studies with experts and students. We discuss how, in a set of solutions, appropriate examples can be automatically identified and assigned to given student solutions via machine learning techniques in conjunction with an underlying dissimilarity metric. The plausibility of such an automatic selection is evaluated in an expert survey, while possible choices for domain-agnostic dissimilarity measures are tested in the context of real solution sets of Java programs. The quantitative evidence suggests that the proposed feedback strategies and

automatic example assignment are viable in principle, further user studies in large-scale learning environments being the subject of future research.

Keywords: example-based feedback; machine learning; intelligent tutoring system; ITS; prototype-based clustering.

Reference to this paper should be made as follows: Gross, S., Mokbel, B., Paassen, B., Hammer, B. and Pinkwart, N. (2014) 'Example-based feedback provision using structured solution spaces', *Int. J. Learning Technology*, Vol. 9, No. 3, pp.248–280.

Biographical notes: Sebastian Gross received his Diploma degree in Business Information Systems from Clausthal University of Technology. After graduating, he joined the research group 'Human-Centered Information Systems' headed by Prof. Pinkwart in October 2011. In July 2013, he followed Prof. Pinkwart to Humboldt-Universität zu Berlin, and is currently working in the research project 'Learning feedback in intelligent tutoring systems' (FIT).

Bassam Mokbel is currently a PhD student at Bielefeld University, in the research group for theoretical computer science within the Center of Excellence for Cognitive Interaction Technology (CITEC). He studied computer science at Clausthal University of Technology, where he received his Diploma degree in 2009, and became a research assistant in the state-funded collaborative research program 'IT ecosystems' about complex and heterogeneous distributed computer systems. He joined the CITEC in April 2010, and is currently working in the context of the DFG priority programme 1527 for 'Autonomous learning'.

Benjamin Paassen received his Bachelor's degree in Cognitive Informatics from Bielefeld University. Since 2012, he has been working as a research assistant in the DFG-funded research project 'Learning feedback in intelligent tutoring systems' (FIT) at Bielefeld University.

Barbara Hammer, after completing her diploma studies in pure mathematics in 1995, received her PhD in Computer Science in 1995 and her Venia Legendi in Computer Science in 2003, both from the University of Osnabrück, Germany. From 2000–2004, she was leading the junior research group 'Learning with Neural Methods in Structured Domains' which was funded within the frame of an innovation initiative of Lower Saxony, before accepting an offer as a Professor for Theoretical Computer Science at Clausthal University of Technology in 2004. Starting from April 2010, she moved to the CITEC Center of Excellence at Bielefeld University as a Professor for Theoretical Computer Science for Cognitive Systems.

Niels Pinkwart has a research background at the intersection of computer science, educational and collaborative technology, and human computer interaction. From 1995 to 1999, he studied computer science and mathematics at the University of Duisburg-Essen, where he also completed his PhD in 2005 as a member of the COLLIDE research team. After a post-doctoral position at the HCI Institute of Carnegie Mellon University (2005/2006), he accepted offers for Assistant Professor and Associate Professor positions at Clausthal University of Technology where he lead the research group 'Human-Centered Information Systems'. In 2013, he moved to

Humboldt-Universität zu Berlin where he heads the research group 'Computer Science Education/Computer Science and Society'.

This paper is a revised and expanded version of a paper entitled 'Feedback provision strategies in intelligent tutoring systems based on clustered solution spaces' presented at 10. e-Learning Fachtagung Informatik (DeLFI), Germany, September 26, 2012.

1 Introduction

Motivation

Intelligent tutoring systems (ITSs) have greatly advanced in recent years. These learning supporting systems aim to provide intelligent, one-on-one, computer-based support to students. Current ITSs typically require a formalised domain model since they need to judge whether and why a given student solution is correct or wrong. Hence, a challenge in ITS research is to extend the applicability of such systems to ill-defined domains which lack a single strong domain theory (Lynch et al., 2010b). Even besides that, a formalisation of the underlying domain knowledge is usually a substantial amount of work even in well-defined domains: researchers have reported 100–1,000 hours of authoring time needed for one hour of instruction (Murray et al., 2003).

Ill-definedness in ITSs

Most of the ITS research and development has been conducted in domains which are characterised by a well-accepted theory or model that makes it possible unambiguously to classify problem solutions as correct or incorrect. Not all domains of teaching and inquiry are that well-defined, indeed most are not. In domains such as law, argumentation, history, art, essay writing or intercultural competence, most problems are ill-defined and have ambiguous solutions that can be argued for (and against!) but that are impossible to verify formally (Voss, 2006; Reitman, 1965; Lynch et al., 2010b).

For those reasons, the notion of ill-definedness presents a number of unique challenges for ITS researchers. While general, transferable system design principles are still lacking, Pople (1982) endorsed the development of systems that support the exploration of alternate problem hypotheses to prevent early student commitment. Jonassen (1997) proposed a problem solving process and a series of pedagogical recommendations including the development of case bases and supporting the construction of knowledge bases that reflect real-world knowledge. Lynch et al. (2006) identified four human tutoring strategies that were applicable for incorporation into ITSs also for ill-defined problems: case studies, collaboration, weak theory scaffolding and expert review. Tutoring systems can be constructed that support these strategies (Lynch et al., 2010a; Aleven et al., 2008, 2007, 2006).

Yet, while some noteworthy attempts at designing ITS systems for ill-defined problems have been made, these are overall currently not as successful as ITSs have shown to be for well-defined ones, and general system design principles are still lacking. One reason for this is that, unlike their electronic counterparts, human tutors *can* give hints in situations where no unique answer exists or where the judgment

of the correctness of a student answer is not clear – which is a characteristic factor of ill-definedness (Voss, 2006; Reitman, 1965). Humans are capable of providing feedback in situations where they do not have explicit full domain knowledge but where they have some limited previous experience or where they have already seen alternative solutions by means of adaptation and re-characterisation strategies (Piaget, 1972; Reitman, 1965; Simon, 1973): they adopt situation dependent representations of possible problem solutions based on examples, and they learn how to compare two solutions and to align their relevant parts. Based on this learned alignment, humans can give feedback about which parts of a new student solution seem wrong in comparison to known examples that are considered as correct.

Approaches for ITSs in ill-defined domains

This observation provides a motivation for designing technical systems which partially mimic human behaviour in such settings by the incorporation of artificial intelligence or data mining techniques. For instance, Fournier-Viger et al. (2010) and Nkambou et al. (2011) proposed a hybrid approach for supporting tutoring services. They combined an expert-system with model-tracing and data mining approaches in order to support learners in operating a robotic arm. Using a cognitive model, main steps for moving the robotic arm were modelled as a set of rules with declarative knowledge. Data mining techniques were used to extract frequently occurring parts from the records of user solutions. Based on these parts, problem-solving steps were suggested to users.

Current research on ITSs for ill-defined problems also includes the work of Ogan et al. (2009) who developed a tutoring system for intercultural competence where students were given an authentic problem in cultural translation, and the system guided them in a clear analysis process consisting of noticing key features, making analyses, receiving feedback, and then reflecting upon their choices. In Walker et al. (2008), models of discussion posts were used to provide feedback to learners. Student solutions were analysed and compared to the model using keyword extraction.

Example-based learning has shown to be effective in supporting learning also in ill-defined domains. In the NavEx tutor, annotated program code examples were provided to students in order to give explanations to learners instead of providing bare solutions (Brusilovsky and Yudelson, 2008).

In this contribution, we investigate the possibility to generate feedback automatically by analysing and exploiting inherent structures in solution spaces using visualisation and clustering techniques known from machine learning (ML). We demonstrate how this approach works independently from the specific domain in which the ITS is applied, and propose suitable feedback strategies relying on example-based learning.

The general idea is that example-based feedback can be generated automatically, as long as a data-driven model is able to assign a suitable example solution to a given student solution. Our approach relies on two prerequisites:

1 The given learning task can be solved in several different ways by students, and from these possibilities a small number of common solution strategies emerges naturally. Although learners are not explicitly restricted to one of these common strategies (e.g., infinitely many solutions might theoretically exist for a task), this is a fair assumption, since typical educational tasks are designed to achieve certain learning goals and are of limited complexity. Therefore, in a sufficiently large

collection of correct student solutions, there would likely exist clusters of similar solutions which implement the same general solution approach.

2 A collection of student solutions is available, including correct/high-quality solutions. Alternatively, a set of designated sample solutions, designed by domain experts to cover common ways to solve the task, is available.

Based on these assumptions, we examine ways to detect and exploit structures in a set of solutions (for ill-defined and well-defined problems). We show how data proximity measures and ML techniques can be utilised for this purpose, and how based on these structures, feedback can be given to students.

Paper overview

First, we discuss feedback strategies based on structured solution spaces in Section 2. Next, in Section 3, we review some relevant existing approaches to apply ML in the domain of ITSs, and we discuss how ML techniques can be used to exploit structures in given solution spaces (independently of a specific domain). As a crucial factor influencing these techniques, we identify the representation of data and the corresponding proximity measures, and we address this topic in detail. The feasibility of the proposed techniques and feedback provision strategies is demonstrated in several evaluation studies from students' as well as experts' perspectives (Section 4). In Section 5, we evaluate several proximity measures in terms of their ability to capture inherent structure in datasets consisting of real student solutions. Finally, we summarise the results and provide an outlook over future research perspectives in Section 6.

2 Feedback strategies based on clustered solution spaces

Typically, feedback in ITSs aims to accompany the learning process, instructing learners in order to support them in finding and correcting mistakes or misconceptions in their solutions. In various studies feedback has demonstrated to play a significant role in instruction (Mory et al., 2004). Research on feedback provision considered aspects such as how feedback messages should be formulated (e.g., response accuracy, correct answer, hints, examples) (Melis, 2005), when a system should intervene to provide feedback (e.g., immediately, or after some time has elapsed) (Kulik and Kulik, 1988), or which pedagogical theory of learning it should be based on. For instance, Zakharov et al. (2005) implemented pedagogical strategies based on the theory of learning from performance errors (Ohlson, 1996) in the EER-Tutor, a constraint-based tutor (Mitrovic et al., 2001) for database design. By re-engineering feedback, they derived principles for how feedback messages should be designed to be effective. Aleven and Koedinger (2002) proposed an effective metacognitive strategy implemented in a geometry cognitive tutor which provides support for guided learning by doing and explaining. In addition to feedback on issues a learner did wrong, positive feedback can help learners to learn better. More recently, research on feedback has refined these aspects. Mitrovic et al. (2013) have shown that providing positive feedback in a constraint-based SQL tutor improves learning at twice the speed than only providing negative feedback on mistakes to learners. In order to help students become better

help-seekers, Roll et al. (2011) examined the use of metacognitive feedback about learners' help-seeking behaviour to avoid their under- or overusage of feedback.

As argued above, one possible way to help learners understand concepts of a specific domain is example-based learning encouraging learners' self-explanation behaviour (Renkl et al., 1998). In our approach, we propose the use of examples to provide feedback in order to help students identify mistakes and misconceptions in their solutions. In our approach, we assume there is a means to identify similarities among a set of solutions. In this case, a newly submitted student solution can be analysed and compared to the existing set of solutions. We further assume that a solution which is highly similar to the student solution can be identified in the existing set. This highly similar solution serves as a *counterpart*, which can then be used for a fine-grained comparison and to provide feedback to a student solution. Here, we distinguish two feedback strategies:

F1 Parts in the student solution which differs partially but implements the same problem solving strategy as its counterpart are *highlighted*, without showing the actual counterpart to the learner.

F2 Parts of the student solution are *contrasted* to parts of its counterpart which implements the same problem solving strategy, without explicitly highlighting differences between the student solution and its counterpart.

The strategies F1 and F2 aim to help learners focus on potential mistakes in their solutions which might require attention by identifying and providing dissimilar parts in otherwise similar solutions from peers. Feedback strategy F1 focuses on guiding learners towards reflecting on their solution and explaining it without showing the counterpart. The learner is asked to explain the highlighted parts in her solution, thus identifying potential mistakes or misconceptions. Feedback strategy F2 requires a learner to understand the contrasted part, to identify the corresponding pieces of her solution, and to compare both parts in order to find a possible mistake. The contrasted part can then directly be used by the learner to improve her solution. Therefore, a learner might transform and adapt the specific characteristics of the counterpart to her solution.

Both feedback strategies can be applied independently of each other, but can also can be combined simultaneously as well as consecutively. Combining both strategies simultaneously supports a learner in identifying similarities by highlighting and contrasting dissimilar parts of her solution and its counterpart, and may thus help the learner focus on specific differences. A consecutive combination of both strategies may be used to adapt the feedback to the learner's needs and progress depending on the strategy which had been applied previously. For instance, applying strategy F2 successfully for a particular learner can lead to applying strategy F1 in future feedback provisions about similar misconceptions. If providing feedback via strategy F1 did not help a learner to improve her solution, strategy F2 can be applied in similar cases of misconceptions.

Figures 1 to 3 illustrate how feedback strategies F1 and F2 can be applied: [F1] Partial differences between the student's solution and the counterpart are highlighted in the student's solution (see Figure 1). [F2] The student's solution is contrasted to the counterpart by showing both side-by-side (see Figure 2). [F1+ F2] In addition, both strategies can be combined simultaneously by contrasting student's solution to the counterpart and highlighting the partial differences in student's solution (see Figure 3).

Figure 1 Highlighting of parts in student's solution

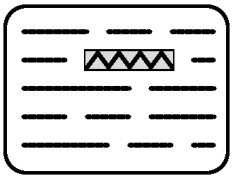
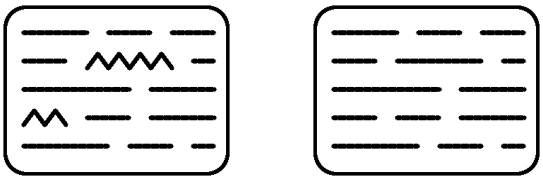
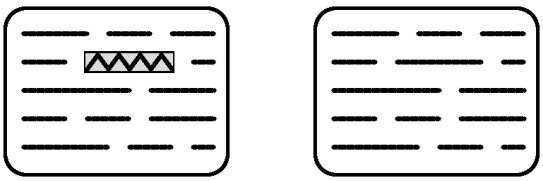


Figure 2 Comparison of solutions, (a) student's solution (b) representative solution



(a) (b)

Figure 3 Comparison of solutions with highlighting of parts, (a) student's solution (b) representative solution



(a) (b)

Level of detail in feedback provision

A crucial aspect in feedback provision is the level of detail of the provided feedback and how to effectively design instructions. For a review on effective instructional explanations in example-based learning, see the work of Wittwer and Renkl (2010). In our approach, feedback must be designed to provide sufficient information about a (potential) mistake to a learner in order to help her to identify and correct this mistake. However, feedback must also guarantee that a learner attempts to reflect on her solution and to explain (potential) mistakes herself. Thus, considering both requirements, we propose to adapt feedback regarding learner's needs and demands. Starting with an initial level of detail of a suitable counterpart, the feedback could be extended successively by larger parts of the counterpart and reduced, respectively, depending on the learners' progress in learning. Moreover, feedback could also be extended according to the learners' demands. Accompanied by meta-cognitive feedback about her help-seeking behaviour as proposed by Roll et al. (2011), such an approach could help learners find the right balance between too less and too much feedback content of a counterpart.

Quality of counterparts

For feedback strategies F1 or F2 being successful, it is crucial that differing parts of two solutions reveal relevant information about a potential misconception or error. Therefore, these strategies seem most suitable in circumstances where solutions are widely semantically or structurally similar (thus aiming at the same overall solution strategy), but differ in few parts only. Thus, the dissimilar parts could be an indication of a mistake or a misconception in the problem solving strategy. Highlighting such potentially erroneous parts or contrasting these parts to corresponding parts of counterparts is a possible way to provide feedback to learners. In a set of student solutions, however, there is usually no guarantee that the counterpart solution is correct (unless, of course, there is a domain model to check this). Here, we distinguish two cases:

1 information about the quality of solutions is (partially) available

2 no information about the quality of solutions is available.

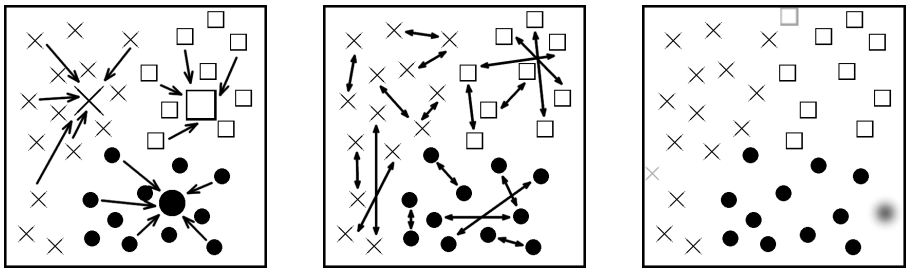
2.1 Feedback provision using representative solutions

In the first case [illustrated in Figure 4(a)], we assume that information about the quality of solutions is (partially) available. This information could, for instance, be based on expert gradings. Then, the solution space consisting of student solutions can be structured so that similar solutions are assigned to the same class. Finally, for each cluster of the solution space a representative (student) solution is identified which

- has a high structural similarity to all other solutions in the cluster (i.e., it is the centre of the cluster)
- has a good quality (i.e., it is a solution which solves the task accurately).

Alternatively, sample solutions implemented by domain experts could be added to the set of student solutions, and assigned to suitable clusters, which then serve as high quality solutions representing a class [shown via large symbols in Figure 4(a)].

Figure 4 Clustering of student solutions, (a) with representative (b) without representative (c) with drop-out



(a) (b) (c)

By applying feedback provision strategies F1 or F2, a student solution is structurally and/or semantically compared to the representative solution of the same cluster and differences between the solutions are highlighted and contrasted, respectively. The consequence of providing such feedback could be that the student solution improves and moves qualitatively towards the representative solution [illustrated in Figure 4(a)].

Design of representative solutions

In order to provide feedback to learners based on representative solutions, a system must be capable to identify a suitable sample for a given student solution. This requires that a set of representative solutions cover typical (and optimal) strategies which solve a given problem. The number of possible and/or valid problem solving strategies depends on the curriculum and the learning goals of a specific domain. For instance, a task of a curriculum in Java programming could ask learners to check if a given string is a palindrome. This task could be solved using loop statements, iterating over single letters or a built-in method of the Java application programming interface (API). Considering the learning goals of the task, a tutor could allow any of the strategies or she could exclude particular strategies. Although the given task is ill-defined since a wide range of possible solutions exist, such learning tasks naturally cause an emergence of a relatively small number of solution strategies, and thus, a limited number of representative solutions to be prepared.

However, it remains a challenge to reliably assign an unseen student solution, for which feedback is requested, to a representative solution. The underlying similarity measure has to be abstract enough to distinguish different solution strategies, but at the same time invariant with respect to individual details in the implementation style. We will discuss these aspects in Section 3.2.

2.2 Feedback provision without representative solutions

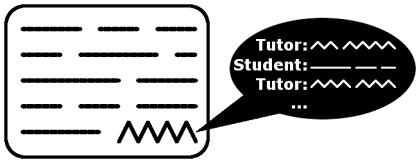
In the second case [illustrated in Figure 4(b)], we assume that no information on the quality of solutions is available. Thus, the solution space can be structured in order to cluster similar solutions in the same class, but representative correct solutions cannot be identified to compare to student solutions. At first sight, this vagueness of correctness seems to be a crucial disadvantage. Yet, with suitable feedback messages accompanying the highlighting or contrasting, this issue can be addressed. Typical messages can be formulated as self reflection prompts which have shown to be an effective form of intervention (Chi et al., 1989). Since a contrasted solution could be erroneous, a message should explicitly point out the vagueness of correctness. For example, students can be asked not only to reflect on their own solution but also on the contrasted solution to identify misconceptions. In the domain of fractions, this approach of learning by erroneous examples has been shown to help students learn from common errors of other students (Tsovaltzi et al., 2010).

Modelled in a procedure of peer reviewing (Topping, 1998), students could also help each other to improve their solutions. Peer reviewing has shown to produce assessment results of good quality as long as four to five reviews for a solution are available (Cho and Schunn, 2007). While peer reviews have the advantage that they build up domain knowledge over time (and thus make the computation of representatives possible), their

disadvantage is that results (and thus, feedback) for a newly submitted solution are usually not available instantly. To avoid this disadvantage, another possible option is peer tutoring (Goodlad and Hirst, 1989; Merrill et al., 1992).

Figure 5 sketches a student solution with a dialogue between student and peer tutor. The peer tutor can recommend a change of erroneous parts of the solution. This feedback can help the student to improve her solution. As a side effect, also the peer tutor gets trained in finding and avoiding mistakes.

Figure 5 Dialogue-based peer tutoring



In this second case, the ITS does not compute feedback directly but acts as a mediator between students. Since the reviewer/peer tutor is a student who is in a learning process herself and does not possess expert knowledge, the ITS has to choose both student and reviewer/peer tutor intelligently. Student and reviewer must be selected in such a way that the student profits by reviewer's tips and the reviewer is not overwhelmed by reviewing student's solution. As examined in Walker et al. (2012), the adaptive peer tutoring assistant (APTA) could help students noticing relevant feedback in order to improve their learning in peer tutoring. The solution structure clustering can be helpful here, since it can guarantee that both students have submitted at least a structurally similar (though possibly not equally good) solution.

2.3 Feedback provision for drop-out solutions

In a third case [shown in Figure 4(c)], we assume that the solution space can be structured but there may be single student solutions (shown via blurred symbols in the figure) which cannot be assigned to a suitable cluster due to a lack of structural or semantic characteristics in the student solution or an odd problem solving strategy which fundamentally differs from strategies represented by the existing classes of the solution space. Here, the purpose of an ITS should be to identify which problem solving strategy the student intended to implement and thus to provide appropriate feedback to the student in order to guide her towards this strategy. In this case, peer tutoring can help a student to firstly identify fundamental misconceptions and mistakes. Alternatively, the student could be presented with all available representative solutions (or if possible abstractions of them) in order to select the most appropriate solution implementing the problem solving strategy the student intended to implement. Based on the selected representative solution, the student can then be guided to successfully implement this strategy using the proposed example-based feedback provision strategies.

3 ML of solution spaces

The discussed feedback provision strategies are based on the assumption that meaningful examples can be identified in a given set of solutions, and that student solutions can

be assigned to these examples in an appropriate manner. This raises the question how to implement these aspects with automatic data-driven methods. In this section, we will discuss how prototype-based ML methods can be utilised for this purpose. We will focus on the important role of data representation techniques and proximity measures, which crucially determine the formation of structures in the solution space.

Let us assume that a set of correct student solutions is given for a learning task. As described above, feedback can then be generated based on a sufficient number of examples, which are essentially high-quality solutions, either included in a database of student solutions, or provided by experts as designated sample solutions. The remaining problems are:

a assigning a given student solution to a suitable example

b identifying prototypical (high-quality) solutions in a database, which can serve as meaningful examples

c identifying adequate groups of students for peer tutoring.

In the following, we will describe how data proximity measures and ML techniques can address these challenges.

Mathematical formalisation

We assume a set of data points $\{\mathbf{x}^1, \ldots, \mathbf{x}^n\}$ is given, where points $\mathbf{x}^i,\ i \in \{1, \ldots, n\}$ refer to student solutions. The points $\{\mathbf{s}^1, \ldots, \mathbf{s}^m\}$ are examples based on which feedback can be generated. Examples either refer to designated sample solutions provided by experts, or to prototypical (high-quality) student solutions in a given database. Since they are, in either case, solutions themselves, they are always specific for the learning task. We require that all student solutions and sample solutions are represented in the same form, and will discuss details about practical data representations later in Section 3.2.

The fundamental mathematical component for our purpose is the notion of proximity: $d(\mathbf{x}^i, \mathbf{x}^j)$ constitutes a fixed measure which assigns a positive value to every pair of data points, indicating their dissimilarity. Without loss of generality, we will use only a dissimilarity or distance measure in the formalisation, instead of a notion of similarity. Similarity values can easily be transferred to dissimilarities, as described in Pekalska and Duin (2005). We use the term proximity for both notions, depending on the context. For example, $d(\mathbf{x}^i, \mathbf{x}^j) = 0$ indicates that two student solutions are equal; a large value indicates that the solutions are very dissimilar. So far, d is just some abstract proximity, we will later discuss different choices in the context of an ITS. For now, we will assume that d evaluates the dissimilarity of a pair of solutions in a desired and meaningful way. Using the proximity measure d, we can address problem (a): for a given student solution $\mathbf{x}^i$, the most similar (and thus possibly most suited) example can be selected simply by

$$\arg\min_j \ d(\mathbf{x}^i, \mathbf{s}^j), \quad j \in \{1, \ldots, m\} \ .$$

To address (b) and (c), various ML techniques can be applied based on the proximity d. We propose prototype-based clustering, since it is directly beneficial to solve problem (b) by finding prototypical points in the data, while also identifying groups in the data to address (c). After the training a prototype-based model, the model represents the data in terms of a small number of k prototypes $\{\mathbf{w}^1, \ldots, \mathbf{w}^k\}$. They decompose the data space into clusters by assigning a data point $\mathbf{x}^i$ to $\mathbf{w}^j$ iff the latter is closest to the data point. Once prototypes are fixed, i.e., the model has been trained, any yet unknown solution $\mathbf{x}^u$ can be assigned to one cluster in the same way. Thus, decisions why a certain element $\mathbf{x}^i$ or any unknown solution $\mathbf{x}^u$ belongs to a given class can be substantiated by referring to this prototype $\mathbf{w}^j$ and examining the respective closest solution. The decomposition of the data space is equivalent to identifying k groups containing very similar solutions, which can be utilised for peer tutoring, i.e., (c). Examples could be selected from the database by considering correct/high-quality student solutions which are very similar to a respective prototype, which addresses problem (b).

So far, we established the theoretical basics of how proximity measures and prototype-based clustering techniques can be applied to solve the three stated problems (a), (b), and (c). In the following, we will discuss in detail some techniques that can be used to implement this in a practical ITS, and how data can be represented. For practical examples, we will refer mainly to a scenario of an ITS for programming.

3.1 Data representation and proximity

Most classical ML methods restrict data formats to finite-dimensional real vectors, i.e., a representation of data in terms of a finite number of descriptive features. This kind of representation is based on capturing meaningful characteristics of the data in separate numerical values, which usually are (complex) statistics about the underlying subjects. The typical way to express the proximity between a pair of feature vectors is to calculate a distance in the underlying vector space, usually the Euclidean distance. While there is ongoing research about feature extraction techniques, it remains hard to integrate highly contextual, compositional aspects in the feature encoding. The question of how compositional aspects can be represented in connectionist approaches has been the subject of an extensive debate (Hammer, 2002). Although low-level representations in terms of vectors provide a high robustness against noise, complex data or data structures require high-dimensional feature vectors for an adequate description. This increases the computational complexity, and, more severely, it often prohibits fast learning due to the curse of dimensionality [which expresses all phenomena that appear with high-dimensional data, and that affect the behaviour and performances of learning algorithms (Verleysen and François, 2005)]. Another disadvantage of vectorial representations is that the more complex and meaningful features are often hand-tailored for a specific problem or domain. This kind of feature selection and construction often constitutes a rather expensive part of the system design, while still underlying principled limitations. To avoid the necessity for domain-specific customisations, ML algorithms must be able to operate on fairly general representations of the solutions in an ITS.

Related topics have recently been addressed in ML research, under the subject of structure learning and *autonomous learning* (Douglas and Sejnowski, 2007): generally, there is a tendency to enhance vectorial data representations which preserve the statistical robustness but enable a high-level integration of structural and functional aspects. Examples for this research include dedicated structure kernels, recursive models,

or metric learners (Hammer and Jain, 2004). Modern data analysis schemes therefore use proximity measures which directly evaluate the (dis)similarity of a pair of subjects, without the need to encode the data in a (possibly inappropriate or inapplicable) feature representation. These measures can take various contextual characteristics into account. Popular applications involve gene sequences (Gusfield, 1997), audio signals (Müller, 2007), texts (Li et al., 2004; Gusfield, 1997), or data from motion capturing (Müller, 2007). Corresponding state-of-the-art ML approaches, including unsupervised clustering and visualisation, as well as supervised learning, rely mainly on some notion of proximity during the training process. Popular examples include *kernel* and *relational* clustering and classification methods (Hasenfuss and Hammer, 2010), as well as nonlinear dimensionality reduction for visualisation (van der Maaten and Hinton, 2008). In the context of ITSs, the question now occurs how proximity measures can be realised practically for the given data $\mathbf{x}^i$, and how the solutions can be represented in a suitable way. Now, we will have a closer look at different possibilities to define proximity measures on such data, referring to different levels of structural complexity from vectorial to graph-based representations.

3.2 Features of proximity measures

As the primary distinguishing feature of proximity measures, we regard the form of representation on which the measures operate, i.e., how solutions are represented to directly apply the measure. These representations can be categorised by their *structural complexity*, i.e., how much structural information they can represent directly:

1 a finite-dimensional feature *vector*

2 a symbolic *sequence*

3 a tree or a *graph* with annotations of nodes and edges.

In the following, we will briefly describe some existing proximity measures in each category. We assume that solutions can be represented in some (formal) language, like, e.g., English essays, computer programs, UML diagrams, etc. Apart from this, we will keep the discussion rather general at this point, but will later formalise the discussed measures with respect to our application scenario for programming classes.

1 To encode a solution as a feature *vector*, one can use standard approaches from classical text and document mining. For example, by extracting the term frequencies within a document (i.e., a solution) in relation to all documents we gain *tf-idf weights* as a feature vector (Salton and Buckley, 1988). Depending on the syntax of the solution, the idea of ‘terms’ can be transferred to any appropriate symbolic or syntactic entity. In general, a feature encoding represents a solution in terms of predefined (possibly complex) statistics, while the order of syntactic elements (or the relationship between them) is usually not (or only implicitly) considered. Therefore, to include structural information it must be encoded within the features explicitly. With a feature vector for every solution, canonical proximity measures are distances in the underlying vector space, most commonly the Euclidean distance, which can be computed efficiently.

2 To consider the proximity based on a *sequence*, we can transfer established string similarity measures to operate on the syntactic elements of the solutions, like the *normalised compression distance* (NCD), or *alignment* measures commonly used in bioinformatics and text mining (Li et al., 2004; Gusfield, 1997). In contrast to a statistical view on the data in form of features, proximity measures for sequences explicitly take the order of symbols into account.

3 The similarity of *graph* or tree structures can be calculated via structure kernels (Neuhaus and Bunke, 2006; Moschitti et al., 2008). Tree kernels are well-established, e.g., for syntax trees in computer linguistics. For instance, to compare a pair of solutions consisting of English text, *natural language processing* (NLP) techniques can derive trees or graphs from the syntax (Moschitti et al., 2008). These proximity measures aim to consider the symbols (usually annotated on the nodes) together with their mutual relationships represented by edges. Here, the performance of the algorithms is a limitation for practical applications. Since it is known that subgraph isomorphism is an NP-complete problem, effective algorithms need to rely on strong restrictions of the problem to guarantee feasible running times (Ullmann, 1976). Therefore, tree kernels are more suitable for large-scale applications than general graph kernels.

The measures we discussed are domain-independent, provided the solutions can be represented in the required form. Since measures from class 3 consider structural relationships, but generally suffer from high computational complexity, they are thus not suited for large-scale applications. We will not consider them in the following experiments, but as a subject of ongoing work. In contrast, measures from class 1 and 2 have feasible running times, and facilitated good results in other ML applications. In Section 5, we provide a quantitative study on their performance in association with clustering and classification techniques on artificial and real student solutions.

Proximity of Java programs

As stated, we assume that solutions are given in some (formal) language. Therefore they can always be transferred to syntax trees, where syntactic elements are annotated nodes, see e.g., NLP approaches (Moschitti et al., 2008). A relational analysis of syntax elements yields additional edges which can create cycles, so that we arrive at a general graph. In the following we assume that solutions are (or could be) represented as graph structures, with different kinds of meta information annotated on the nodes and edges. Each node represents a (syntactic or semantical) element of the solution, and edges establish relationships between these elements. This is a very general format, which is able to represent a variety of different data, including solutions in existing ITSs, e.g., argumentation diagrams (Loll and Pinkwart, 2013).

In our application scenario, we consider *syntax trees* of Java programs. We used a parser (the official Oracle Java Compiler API) to derive a syntax tree from the Java source code of a given solution. Solutions $\mathbf{x}^i, i = \{1, \ldots, n\}$ are thus trees $G_i = (V_i, E_i)$ comprised of sets of vertices V_i and edges E_i. All vertices are attributed to a certain node type or label, which is a symbol from the finite alphabet $\Sigma = \{l_1, \ldots, l_T\}$, by a relation $R \subset V \times \Sigma$. The nodes represent syntactic units of the program and their types represent a general category like the declaration of a variable, a function call, a

logical expression, a variable assignment, etc. The alphabet is defined by the parser and is therefore the same for all solutions in the set.

We can derive tf-idf weights as vectors $\mathbf{t}^i \in \mathbb{R}^T$ using the classic way described, e.g., in Salton and Buckley (1988), regarding the term frequency as the frequency of the respective symbol, and each syntax tree as a document. Based on the vectors $\mathbf{t}^i, \mathbf{t}^j$, we define the distance between the respective pair of solutions as the Euclidean distance:

$$d_{\texttt{tfidf}}(\mathbf{x}^i, \mathbf{x}^j) = \sqrt{\sum_{r=1}^{T} \left([\mathbf{t}^i]_r - [\mathbf{t}^j]_r\right)^2} \ .$$

This feature encoding captures statistics about the symbols, however their ordering (and other relations between them) are not considered.

In contrast, alignment measures take into account the sequential ordering of symbols. For this purpose, we encode the syntax trees as sequences $\mathbf{q}^i \in \Sigma^*$ by visiting vertices in a depth-first-search order, and concatenating the annotated symbols. This ordering corresponds to the original sequence of the statements in the Java source code which are represented by the nodes. A global alignment of two such sequences $\mathbf{q}^1$ and $\mathbf{q}^2$ consists of an extension of either sequence to $\bar{\mathbf{q}}^1$ and $\bar{\mathbf{q}}^2 \in (\Sigma \cup \{\_\})^*$ where $\_$ denotes a gap, such that $\bar{\mathbf{q}}^1$ and $\bar{\mathbf{q}}^2$ have equal length. This allows us to accumulate costs by the sum

$$\sum_{i=r}^{|\bar{\mathbf{q}}^1|} d\left([\bar{\mathbf{q}}^1]_r, [\bar{\mathbf{q}}^2]_r\right)$$

where d refers to some pairwise similarity function for symbols from $\Sigma \cup \{\_\}$, e.g., a simple delta function. The proximity of two sequences $\mathbf{q}^1$ and $\mathbf{q}^2$ can then be defined as maximum proximity taken over all possible global alignments. Alternatively, if noise is present, one can rely on *local alignment* which can disregard parts of the sequences if they do not match, meaning that the optimum is taken over all alignments of all possible subsequences of the sequences $\mathbf{q}^1$ and $\mathbf{q}^2$. In either case, dynamic programming enables an efficient computation of these terms (Gusfield, 1997). For two solutions $\mathbf{q}^i, \mathbf{q}^j$, we refer to the overall similarity score of a local alignment by $a(\mathbf{q}^i, \mathbf{q}^j)$. For further convenience and consistency, we invert the similarity as proposed in Pekalska and Duin (2005) to obtain the dissimilarity value

$$d_{\texttt{align}}(\mathbf{x}^i, \mathbf{x}^j) = a(\mathbf{q}^i, \mathbf{q}^i) + a(\mathbf{q}^j, \mathbf{q}^j) - 2a(\mathbf{q}^i, \mathbf{q}^j) \ .$$

The third proximity measure that we consider is the *normalised compression distance* (NCD), which also operates on arbitrary symbolic strings (Li et al., 2004). The NCD is based on an approximation of the Kolmogorov complexity and compares two strings by referring to their compressed sizes as given by a real world compressor, like (in our case) the Lempel-Ziv-Markov chain compressor (LZMA). Considering the solutions as sequences (in the same way as for local alignment), we refer to the compressed size of a sequence $\mathbf{q}^i$ as $z(\mathbf{q}^i)$, and denote the concatenation of two sequences as $\mathbf{q}^i\mathbf{q}^j$. Then, the NCD is defined as:

$$d_{\texttt{NCD}}(\mathbf{x}^i, \mathbf{x}^j) = \frac{z(\mathbf{q}^i\mathbf{q}^j) - \min\{z(\mathbf{q}^i), z(\mathbf{q}^j)\}}{\max\{z(\mathbf{q}^i), z(\mathbf{q}^j)\}} \ .$$

Syntax, structure, and semantics of Java solutions

The three dissimilarity measures as discussed above deal with different structural levels of the program, taking into account different structural invariances. In this context, structural invariance means that some changes of the syntax do not affect the result of the dissimilarity measure. For example, to some extent the NCD measure can cope with the permutation of program blocks, while an alignment is always bound to the sequential ordering of the input sequences. On an abstract level, it would be desirable if these invariances corresponded to semantic invariances of the program, where semantics refers to the actual algorithm implemented.

In case of the Java language (like most programming languages), a particular source code (i.e., the structured syntax) produces only one semantic result, whereas a particular result can be generated by various incarnations of code. Following this fact, there exist syntactic invariances w.r.t. semantics, for example, a simple syntactic invariance would be the renaming of variables in the code, or the arbitrary ordering of parts of the program which can be executed concurrently and independently of each other. These kinds of invariances are merely special instances of a generally nonlinear relationship between syntax and semantics, meaning that small variations of the syntax may lead to drastic semantic changes and vice versa. The dissimilarity measures as proposed above do not explicitly take this into account, being based on a domain-agnostic premise in the first place, albeit some of these invariances can be captured by the above-introduced measures.

Not only in the Java domain, the described nonlinear characteristics pose a challenge regarding feedback provision, depending on the given learning assignment or task. We note that it is not necessary to restrict to only one dissimilarity measure, rather they can be regarded in parallel, or they can be applied in a hierarchical fashion, thereby realising different invariances at different levels of detail.

3.3 Prototype-based clustering

Having defined suitable dissimilarity measures for solutions, we will now focus on ML techniques which extract and exploit structures in a solution space based on these dissimilarities. In literature, clustering is a common tool for the inference of structure from (Java) programs. Typically, the focus lies on the structuring of a single source code file, package, or project only, see e.g., Bailey et al. (2012) and Kuhn et al. (2007), or a direct feature description of the program or its semantics is used, the latter requiring domain-specific knowledge (Striewe and Goedicke, 2011, 2013). In contrast, we are interested in clustering techniques based on domain-agnostic pairwise dissimilarity measures as introduced above.

Many different ML techniques are currently available to infer representative prototypes given a set of data points (Kohonen, 1995). Most techniques have been proposed for data represented by Euclidean vectors, which makes them unsuitable for our purposes. Relational Neural Gas (RNG) constitutes one extension to non-Euclidean dissimilarities (Hasenfuss and Hammer, 2010; Hammer and Hasenfuss, 2007) of the form $d(\mathbf{x}^i, \mathbf{x}^j)$ where we assume symmetry ($d(\mathbf{x}^i, \mathbf{x}^j) = d(\mathbf{x}^j, \mathbf{x}^i)$) and normalisation of the self-dissimilarities ($d(\mathbf{x}^i, \mathbf{x}^i) = 0$). It aims at finding prototypes which are as representative for their cluster as possible. This aim is formalised mathematically by the objective to place prototypes at positions such that the average

dissimilarity within clusters is minimal. Thereby, prototypes are represented implicitly as linear combinations $\mathbf{w}^j = \sum_i \alpha_{ji}\mathbf{x}^i$ with $\sum_i \alpha_{ji} = 1$. It was shown that any distance calculation between the 'artificial' prototypes and 'real' solutions can be evaluated implicitly, based on the known dissimilarities only. Therefore, standard numeric methods can be used to optimise the objective function. A number k of representative prototypes of the form $\mathbf{w}^j = \sum_i \alpha_{ji}\mathbf{x}^i$ results this way, although their explicit form is never referred to during training, since the vectorial representation of the input data remains unknown. Instead, the explicit prototypes can be approximated by the respective closest explicit solution(s) and utilised for example-based feedback strategies. From a pedagogical point of view, depending on the scenario, one might also consider to select the closest high-graded or high-quality solution as representative samples.

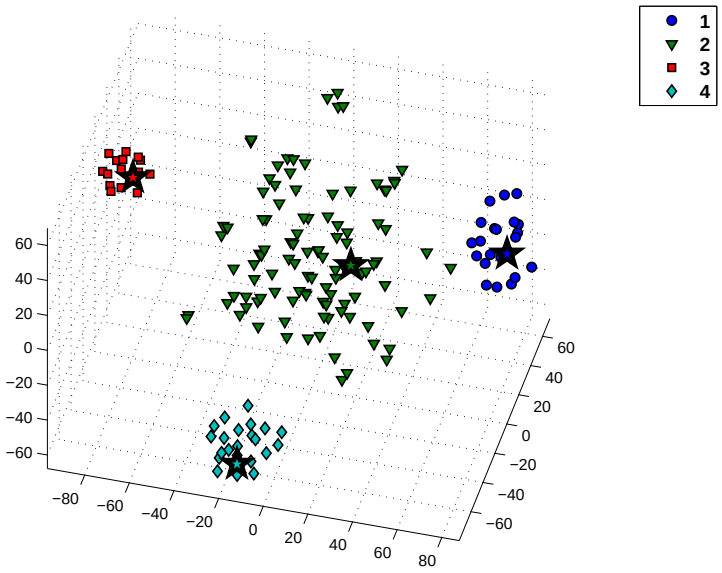
3.4 Preliminary evaluation

For validating our approach of automated feedback provision based on clustered solution spaces in a preliminary test study, we applied the RNG clustering algorithm to a small real world dataset. Our goal was to estimate the potential of unsupervised ML to analyse the solution space of a real ITS for programming courses. Three data proximity measures and data representations were used in the process. Our evaluation was performed using a set of solutions from a Java programming class (first semester students) at Clausthal University consisting of 165 student solutions, where the programming assignment was to create a function which decides for any given input sentence whether palindromes or pangrams are contained. A grade assigned by a human expert was available for every student solution. For a more detailed description of the dataset, see Section 4.2.

We extracted a descriptive string representation from each given Java source code, using the plagiarism detection software Plaggie (Ahtiainen et al., 2006). This results in a string signature for every solution, which is a sequence of symbols, each encoding the node types appearing in the syntax tree. The procedure is similar to the one we describe in Section 3.2. Three different proximity measures were applied based on the string signatures of the solutions, each resulting in a matrix D of pairwise dissimilarities. To investigate which measure is most suited to infer meaningful clusters, we created 3-dimensional visualisations of each dissimilarity dataset, using the state-of-the-art dimensionality reduction technique *t-distributed stochastic neighbour embedding* (t-SNE) (van der Maaten and Hinton, 2008).

One of the measures (greedy string tiling) is a specialised proximity measure for plagiarism detection in documents, and has also been used for Java programs (Ahtiainen et al., 2006). For this measure, the visualisation of the pairwise proximities clearly showed a structure of four clusters, which was the most distinctive and noticeable pattern observed in the experiment. Therefore, we performed a clustering by RNG based on these proximities, fixing the number of prototypes to four, with regard to the articulate 4-cluster structure in the visualisation. Figure 6 shows a t-SNE visualisation of the data, clustered by RNG. The color of the mapped data points marks their cluster assignment found by RNG, which clearly is consistent with the 4-cluster structure. The highlighted points have been identified as examples in the set of student solutions, by picking the closest high-graded solution to the respective prototype. In a small expert survey, the plausibility of this automatic choice was evaluated (see Section 4.1).

Figure 6 The figure shows a visualisation of proximities for Java programs for an early evaluation using a proximity measure from Ahtiainen et al. (2006), specialised for plagiarism detection (see online version for colours)



Note: The different colors for the data points show the result of RNG clustering, where four examples are identified within high-quality student solutions (the large symbols).

4 Evaluation of feedback provision strategies

Next, we investigated the feasibility of feedback strategies as described in Section 2 with datasets of introductory programming courses and conducted a study in a class where students were taught programming in Java. First, we conducted an expert evaluation with four experts to investigate the quality of the automatic clustering and example selection. Furthermore, we examined the effectiveness of the proposed feedback provision from an expert's point of view. The experts were highly experienced and fully qualified to check and assess Java programs due to their long experience as human tutors in Java programming courses.

4.1 Expert survey

To determine the effectiveness of the proposed feedback provision strategies we examined the following hypotheses:

H1 Comparing a student solution to a representative solution helps a student to improve her solution if

- a feedback provision strategy F1 is applied without explicitly showing the representative solution

b feedback provision strategy F2 is applied without explicitly highlighting potentially erroneous parts in the student solution.

H2 Clustering with RNG (see Section 3.4) assigns student solutions to suitable clusters represented by a solution with a high similarity to each solution within the cluster.

Hypothesis H1a was evaluated by asking the four human experts whether or not the proposed feedback strategy F1 is appropriate in order to help the student to improve her solution. Hypotheses H1b and H2 were evaluated by asking the human experts whether or not a representative solution is suitable for a direct comparison. We based our examination on a solution space (see Figure 6) which was clustered using RNG (see Section 3.4), expecting that each cluster consists of student solutions with a high similarity to each other. The structured solution space was then used to select a student solution, a suitable representative solution of the same cluster and a second representative solution of a different cluster.

Each of the four human tutors was instructed to check and assess student solutions by an evaluation sheet. The evaluation sheet contained three questions for every student solution:

1 How do you rate the solution?

2 Which of the proposed representative solutions is suitable for a direct comparison?

3 How useful is the highlighting of potentially erroneous parts for improving the student's solution?

The first question should be answered on a scale from 1 (very poor) to 5 (very good). For the second question, the experts could select from four options:

1 the representative solution for the cluster as found by RNG (described in Section 3.4)

2 a randomly selected solution of the other clusters

3 both solutions

4 none of them.

The third question should also be answered on a scale from 1 (not useful) to 5 (useful).

Two feedback provision strategies were examined in this evaluation. We were interested in whether or not the provision of feedback by comparing to a representative solution can potentially help students to improve their solutions. The results implied that the appropriate feedback provision strategy depends on the rating of the solution. Thus, all analyses were performed considering the quality of the student solution. The experts were asked to give a score on a five-point scale (1 = not useful, 5 = very useful) rating how useful feedback strategy F1 is (see Table 1). Furthermore, they should select a representative solution which is suitable for applying feedback strategy F2 (see Table 2).

The poorer the rating of the solution, the better the feedback strategy F1 was assessed by the experts. For very poor solutions, the feedback strategy was highly rated with an average of 4.83. For solutions that already have a good quality, a comparison

to a representative solution does not make much sense, because the solution cannot be improved.

Table 1 Average score for feedback provision strategy F1

n	*rating*	*mean*	*sd*	*median*
12	Very poor	4.83	0.389	5
15	Poor	4.20	1.082	5
16	Average	3.44	1.315	4
40	Good	3.1	1.355	3
78	Very good	1.95	1.247	1

Table 2 Selected options for feedback provision strategy F2

n	*Rating*	*Representative solution*		*Both (3)*	*None (4)*
		Of the same cluster (1)	*Of a different cluster (2)*		
12	1	0	3	9	0
15	2	4	4	7	0
16	3	1	7	5	3
40	4	15	11	6	8
78	5	11	15	11	41
161	-	31	40	38	52

The evaluation indicated that the experts did not rate a comparison of a (very) good solution to a representative solution as valuable in most cases. However, they stated that for (very) poor solutions, a comparison to both representative solutions would be educationally beneficial.

The evaluation confirmed our expectations that the feedback provision strategies proposed in Section 2 can help a student to improve her solution. For solutions which are assessed as (very) poor, highlighting potentially erroneous parts in the student's solution makes sense. Also, for (very) poor solutions, representative solutions are suitable for a *direct comparison*.

The result of the clustering algorithm did not assign student solutions to suitable clusters unambiguously. The four experts identified the closest cluster representative in 31 out of 70 cases only. This can be traced back to two possible reasons: the structure inferred by the clustering algorithm or, alternatively, the given task could be too crude such that the found representative prototypes do not yet allow for a sufficient level of detail, resulting in a different assignment of solutions to prototypes by experts. This possibility is substantiated by the fact that two different solutions were ranked as suitable by experts in many cases. Alternatively, The chosen metric based on which the clustering is made can be yet unsuited to accurately mirror the relevant semantics of the considered programming task. Nevertheless, it has been demonstrated unambiguously that feedback provision strategies based on similarity of solutions to known cases and, in particular, feedback by highlighting differences were considered as a suitable strategy to deal with incorrect student solutions.

4.2 Expert clustering

In the previous evaluation, we asked four human tutors of a Java programming course to decide whether or not feedback provision strategies as proposed in Section 2 can help students to improve their solutions. As a result of this survey we determined that clustering using RNG did not assign student solutions to suitable clusters unambiguously. It might be the case that the metric used to calculate the pairwise similarity of solutions did not cover appropriately semantic and structural characteristics of Java programs. In order to understand which characteristics of Java programs are distinguishing features and to build a ground truth for evaluating new clustering approaches, two human experts clustered a set of student solutions manually. The solution set consisted of 107 Java programs which had been implemented by students of an introductory programming course and had been graded by human experts.

In the task, the students were asked to implement a Java program which checks whether or not a given text

1 is a palindrome

2 contains a palindrome

3 is a pangram.

The human tutors detected two different strategies for each subtask (1–3) (see Table 3). They clustered the solutions using a three-bit coding, one bit for each subtask representing one of the two strategies. Thus, the set of student solutions were structured in eight clusters and an additional cluster for solutions which could not be assigned to a suitable cluster due to the fact that the experts could not identify a comprehensible problem solving strategy in a specific student solution caused by a lack of semantic and/or structural meaning.

Table 3 Implemented strategies in student solutions as detected by human experts

#	*(1) is palindrome*	*(2) has palindrome*	*(3) is pangram*
1	Reverses string using built-in function or loop and checks whether or not origin string and reversed string are equal	Splits string into words using built-in function and checks words using implemented strategy for subtask (1)	Checks each letter from a to z using a loop
2	Compares letters from outer inwards	Splits string using loop and checks words using implemented strategy for subtask (1)	Checks each letter from a to z manually

The combination of one of the two strategies for each of the three subtasks built the coding for the clusters. Table 4 shows the coding for each of the eight strategies and an additional cluster for non-assignable solutions and its corresponding cluster size.

What lessons do we learn from the expert clustering? It is noticeable that the experts considered both structural as well as semantic characteristics of solutions to cluster solutions. In subtasks (1) and (2), the experts abstracted from structural variations to

similar semantic strategies, whereas in subtask (3) the experts distinguished between structural variances of a similar semantic approach. Furthermore, we can determine that a solution of a specific task may consist of several subtask-specific strategies which may also be implemented in solutions of other classes. That implies that a fine-grained comparison of substructures in solutions could also include solutions of other classes. For instance, strategy one of subtask (1) can be found in clusters coded by 111, 112, 121, and 122. Thus, a representative solution of each of these clusters could be used to provide feedback (as described in Section 2) to a student solution which is potentially erroneous in the specific subtask.

Table 4 Strategies and cluster sizes

Coding	111	112	121	122	211	212	221	222	-
Number of solutions	31	2	3	2	19	5	8	5	32

4.3 Laboratory study

In a first expert survey (see Section 4.1), we asked experts whether the provision of feedback as described in Section 2 can help a student to improve her solution. The evaluation confirmed our expectation that example-based feedback provision makes sense from experts' point of view. Next, in a small lab study, we conducted a Wizard of Oz study to determine how the provision of feedback affected student solutions. Therefore, we observed the effect of feedback on changes in student solutions and their quality.

For the evaluation, we used a web-based programming environment enabling students to write, compile and execute program code written in Java. The code editor supported syntax highlighting as known from popular development environments such as Eclipse or Netbeans. In addition to standard error messages generated by the Java compiler and interpreter, students were able to request feedback to their solutions. They were informed that an ITS would generate this feedback. However, a human expert generated the feedback on request using a web-based tutor interface and provided it to the student interface. The human tutor was able to select parts of both the student solution and a suitable solution representing the cluster student's solution belongs to. For this purpose, a set of sample solutions implementing the most common problem solving strategies for the given task was prepared in advance.

The study was conducted, separated into two parts, by presenting a video lecture with slide show about repeat statements in Java, and a following task where students were asked to implement a Java program (involving repeat statements) using the web based programming environment.

While conducting the exercise, students were allowed to request feedback as often as they wanted, to use their documents and records about the course, and to gather information using the internet. The human tutor (an experienced Java tutor) generated and provided feedback immediately on request, processing a queue with feedback requests. Applying strategy F2 means that a student has to match a contrasted part to an appropriate part in her solution, which might have been difficult in our setting (1st semester non-computer science students being introduced to Java programming). We therefore decided to apply strategy F2 simultaneously with strategy F1. Thus, we tested

both strategy F1 and the combination F1+F2 in the experiments. The human expert was free to choose which one to provide upon a specific request, taking into account learner's needs.

The study was attended by five students, four of them requested feedback at least once. Overall feedback was requested 17 times. Due to the small test group and the lack of a control group, we did not statistically analyse the results but performed a qualitative analysis of student solutions. The human expert, who also provided feedback during the study, assessed each single solution step of each participant when the student requested feedback (*R*), and after one of the feedback provision strategies (F1 or F1+F2) was applied. In Table 5, we show an example how a student solution improved over time by providing feedback.

Table 5 Feedback requests and provision and expert's rating over time

Time	0:00	5:49	8:59	14:19	19:48	22:08	23:18	26:47	32:21
Action	*R*	F1, *R*	F1+F2	*R*	F1+F2, *R*	F1+F2	*R*	F1+F2, *R*	-
Score	2	2	2	3	4	4	4.5	5	5.5

On a scale from 0 to 8 points, the students finally achieved, on average, 7.1 points (sd = 1.084). The expert's rating showed that the the student solutions improved over time (as shown in Table 5). The qualitative analysis of the student solutions revealed that, in some cases, the improvement can be directly explained by the provision of feedback. This is supported by students' responses. The four students were asked on a scale of 1 (not useful) to 5 (very useful) how the provision of feedback helped them to improve their solutions. On average they gave a rating of 4.25 (sd = 0.9574, median = 4.5).

5 Quantitative comparison of proximity measures for clustering solutions

For the following experiments, we use five datasets consisting of Java programs. One dataset was constructed by a tutoring expert as a test benchmark, and is comprised of deliberately designed solutions which emulate student solutions. The other four datasets contain real student solutions, collected in the context of programming courses for business students at Clausthal University. Each individual solution in a set comes from a distinct student and was graded by an expert. In every dataset, we parse the given Java programs and preprocess the syntax trees as described in Section 3.2. Based on the tree representation, we used three different proximity measures in total:

- *tf-idf*: the Euclidean distance $d_{\texttt{tfidf}}$ between the tf-idf weight vectors
- *Alignment*: dissimilarities $d_{\texttt{align}}$ obtained via local alignment by the Smith-Waterman algorithm on the dfs-sequences of node types
- *NCD*: dissimilarities $d_{\texttt{NCD}}$ from the normalised compression distance between the dfs-sequences of node types.

5.1 Dataset descriptions

Artificial data: As a first testbed, we use an artificially created dataset, with 48 programs deliberately written to contain certain characteristics. The programs were created by a human tutoring expert, who was asked to implement different strategies to solve the following overall task: For a given input sentence (i.e., a sequence of strings separated by whitespaces), decide whether all words are palindromes. 8 different strategies emerged from the combination of three design choices which the expert used in the programs, which are inspired by the outcome of the survey discussed in Section 4.2:

1 Does the solution use Java utility functions or only own methods based on primitive language concepts?

2 Does the solution consider the input to be of type String or an array of characters?

3 Does the solution split the sentence into words first and then decide whether the single words are palindromes, or does it iterate over the input sentence as a whole?

Therefore, we can address each of the eight algorithmic approaches by a 3-bit code, which we will refer to in the following by a code of three letters A or B referring to the above order, e.g., ABB. For each of the eight strategies, the expert implemented a prototypical solution (numbered 0 in our experiments), along with five slight variations (numbered 1–5). These six solutions for each strategy simulate six different students using the same overall approach, but with subtle modifications in the details. The six programs are thus mostly equal on an algorithmic level, but are different in programming style (more comments, different names of methods and variables). Each respective variant number 5 contains stronger modifications which slightly alter the algorithmic trace (swapped method declarations or operations, and additional lines of code, e.g., for storing interim data). In total, the dataset therefore consists 48 solutions in eight subsets, each with six solutions, where a distinct cluster structure is expected. Thus, we have a clear predefined structure in this dataset, which consists of eight clusters. A two-dimensional embedding of the proximities by d_{NCD} is shown in Figure 7.

'Newton' data: These are 144 real student solutions, where the learning task was to numerically determine the zero of a given 2nd order polynomial by Newton's method.

'Tax' data: The 155 real student solutions in this set calculate the income tax according to American standards for a given income profile.

'TextCheck' data: This set consists of 68 student solutions taken from the expert clustering described in Section 4.2. We were only able to use 68 out of the total 107 solutions, since we had to rely on syntactically correct programs which could pass the necessary parsing and preprocessing without any failure. The experts categorised the solutions according to eight possible strategies of which one is not included in our reduced set, so our data is comprised of seven classes in total. In general, the solutions are very heterogeneous and the classes are highly imbalanced, some consisting of only two or three solutions. Note that the students are beginner-level programmers, and are not involved in computer-science-focused curricula. Therefore, some solutions

implement rather unusual behaviour, such as replacing upper case letters by lower case letters with distinct if-clauses for every letter in the alphabet, sometimes even nested if-else statements. These tough conditions demonstrate the ill-defined characteristics of the domain and pose a challenge for domain-independent proximity measures.

'Tasks' data: This set consists of 438 real student solutions, and is the joint set of the three previous real datasets, i.e., *'Newton'* (144 solutions), *'Tax'* (155 solutions), and an extended version of the *'TextCheck'* with 139 solutions. With a meaningful proximity measure, we expect that this data would show a clear structural separation into three clusters, which might contain further sub-clusters. A two-dimensional embedding of the proximities by d_{NCD} is shown in Figure 8.

5.2 Results for cluster validity

To evaluate possible choices for proximity measures in a quantitative fashion, we assess the 'distinctiveness' of the clusters as they are obtained by a RNG clustering. We used the well-established *Davies-Bouldin* cluster validity index (Liu et al., 2010), which indicates how well the given clusters are separated from each other and form tight groups in themselves. It evaluates a ratio of the distances within clusters (i.e., cluster 'density') versus the distances between cluster centres (i.e., their overall 'spread'), which ultimately gives a quantitative assessment about the spatial separability or distinctiveness of the given clustering. This evaluation has the practical advantage that it does not rely on ground truth about factually meaningful clusters in the data, information which can only be gathered from human experts specific to the domain and the task. Instead, we can only judge which of the measures leads to the most distinct and less ambiguous cluster structure in association with RNG clustering.

Figure 7 2D t-SNE embedding of the pairwise proximities d_{NCD} for the *Artificial* dataset (see online version for colours)

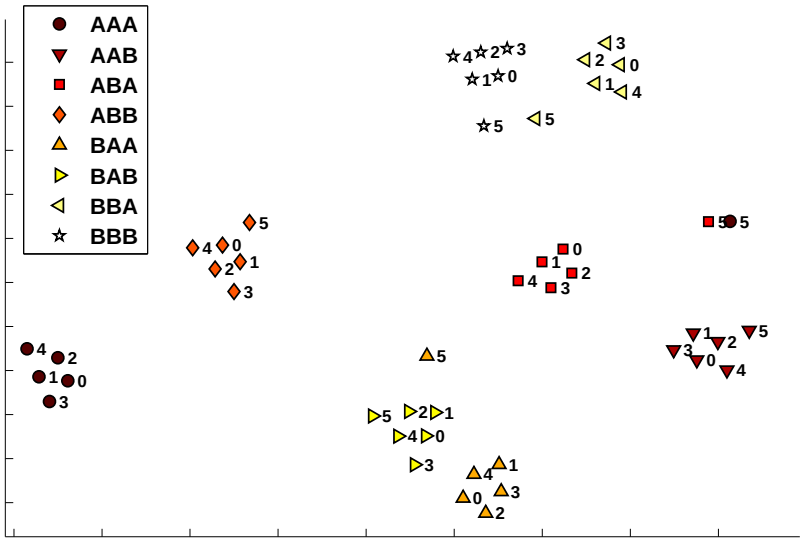
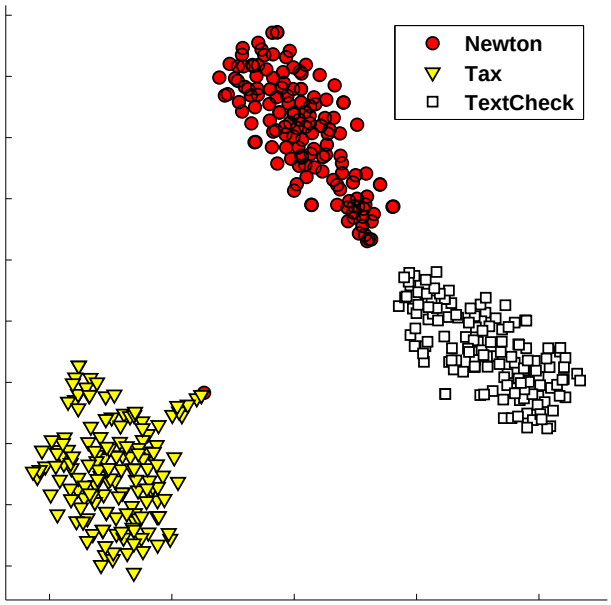


Figure 8 2D t-SNE embedding of the pairwise proximities $d_{\texttt{NCD}}$ for the *'Tasks'* dataset (see online version for colours)



In our evaluation, the pairwise proximities of all solutions were calculated for each dataset by the three mentioned proximity measures: $d_{\texttt{tfidf}}$, $d_{\texttt{NCD}}$, $d_{\texttt{align}}$. The respective proximities serve as input for subsequent clustering by RNG, in which we trained the model using different numbers of prototypes $k = \{2, \ldots, 8\}$. The resulting cluster assignments are evaluated by the Davies-Bouldin index. Table 6 6 shows, for each dataset and proximity measure, the results of the *Davies-Bouldin* cluster validity measure, where smaller values are better. The respective clusterings for the evaluation were obtained with RNG using different numbers of prototypes $k = \{2, \ldots, 8\}$. Each table entry states the respective best of those results, and the corresponding number of prototypes is given in braces. Since the values are hard to interpret on their own, the number printed in italics below serves as a reference. It states the outcome of the Davies-Bouldin measure if the data is assigned randomly to k clusters (instead of the given clustering by RNG with k prototypes). Since a random assignment does not adhere to the structure present in the data, it reflects upon a worst case value. The results show that local alignment dissimilarities $d_{\texttt{align}}$ and tf-idf distances $d_{\texttt{tfidf}}$ had the best overall ratings with respect to the corresponding reference value. This indicates that these measures exhibit the most distinct structures in the data. Looking at the number of clusters for the respective best results, we can generally observe that they are either at the high end of the given spectrum of choices, with $k = \{7, 8\}$, or at the low end, with $k = \{2, 3\}$. Intermediate settings for $k = \{4, 5, 6\}$ were thus rated lower, which means that partitionings of the data were generally more distinct for smaller or larger k. This seems plausible, since we can expect a hierarchical nature of clusters in the data: on the one hand, there are few major strategies to solve the given learning tasks which yield a small number of fairly big clusters; on the other hand there are smaller design choices within the major strategies which would form sub-clusters. Depending on the proximity

measure, separating either the large- or the small-scale structures yields the best result in the validity index.

For the *Artificial* and the *TextCheck* data, we know a semantically plausible cluster separation, since the solutions were categorised by human experts. The selected best number of clusters in the table sometimes coincides with this known categorisation. In case of the *Artificial* data together with the measures $d_{\texttt{NCD}}$ and $d_{\texttt{tfidf}}$, the best value was achieved by RNG clustering with eight prototypes, which corresponds to the actual number of solution strategies in the data, see the data description above. The alignment measure $d_{\texttt{align}}$ comes close with $k = 7$. From the expert survey conducted on the *TextCheck* dataset, described in Section 4.2, we know that eight solution strategies were identified. However, for technical reasons the solutions for one of the classes could not be parsed, therefore the best index for the $d_{\texttt{NCD}}$ measure yields the correct number of clusters in the data, with 7. The *Tasks* dataset is comprised of three different learning tasks, wherefore we expect the most distinctive clustering at $k = 3$. However, the best index for all proximity measures was achieved with two clusters.

Table 6 Results of the *Davies-Bouldin* cluster validity measure

Dataset name	$d_{\texttt{align}}$	$d_{\texttt{NCD}}$	$d_{\texttt{tfidf}}$
Artificial	0.40 (7)	0.67 (8)	0.36 (8)
	3.46	*2.34*	*3.33*
Newton	1.00 (2)	1.75 (2)	1.22 (8)
	1.37	*2.03*	*2.51*
Tax	0.80 (3)	2.08 (8)	0.83 (8)
	1.59	*2.81*	*2.71*
TextCheck	0.92 (2)	1.96 (7)	1.10 (2)
	1.34	*2.70*	*1.40*
Tasks	0.71 (2)	1.06 (2)	1.53 (2)
	1.29	*1.49*	*1.73*

Note: To avoid good assignments by coincidence, we took the mean over ten repeats using a different random assignment each time.

5.3 Results for nearest neighbour classification

While the previous evaluation gives us a vague indication of the plausibility of our approach, we next examine the performance of the proximity measures in the presence of predefined cluster structures. For the *Artificial* and the *TextCheck* data, a semantically plausible classification of each solution is given in terms of a certain solution strategy that it implements. The *Tasks* data is comprised of solutions for three different learning tasks, so we know the respective task to which a solution belongs, and we can expect an articulate cluster structure according to these three groups. For these datasets, we can therefore evaluate classification techniques and compare the different proximity measures. Every data point $\mathbf{x}^i$ is attributed a class label c^i which identifies its class membership by a nominal value. We use a simple *k-nearest-neighbour* classifier (k-NN), in which the classification model relies only on known pairwise proximities, without the need to train a model. There are more sophisticated classification algorithms in the ML literature, but we restrict our analysis to this simple technique to focus

our comparison on the characteristics of the proximity measures only. The resulting classification accuracies for a 3-NN are reported in Table 7. For the *Artificial* and *Tasks* data, accuracies are generally high. This shows that all proximity measures are able to capture the (rather articulate) structure in the data distributions. In the latter case, the three learning tasks from which the solutions originate can be distinguished accurately, which is expected from a meaningful proximity measure. The *Artificial* data poses a slightly harder challenge since the eight groups of solution strategies are distinguished in by rather subtle changes in the original Java source code. Here, the NCD shows a slightly better performance than the other two measures. The *TextCheck* data contains the most challenging class structure, since the real student solutions are heterogeneous and the classes are highly imbalanced. Still, the classification by a simple 3-NN can identify over 40% of the data points accurately, in case of the proximities from NCD and local alignment.

Table 7 Classification accuracies of a 3-NN classifier with different proximity measures on the three datasets where predefined class labels were given

Dataset name	$d_{\texttt{align}}$	$d_{\texttt{NCD}}$	$d_{\texttt{tfidf}}$
Artificial	0.94	1.00	0.94
Tasks	0.98	1.00	0.98
TextCheck	0.41	0.43	0.34

6 Conclusions and outlook

Learning increasingly takes place in environments where human tutors are not able to accompany the learning process directly. For example, massive open online courses (MOOCs) support the distributed learning process via the internet but also bring new challenges (Kop, 2011). These technological developments result in a demand of automatisms which are able to support learning independently from a tutor.

In this paper, we have demonstrated a possible way to automatically provide ITS feedback in the absence of explicit formal domain models, thereby circumventing the need to tailor an ITS to a particular domain or task. Our domain-agnostic approach is based on example-based feedback strategies, which rely solely upon comparing the learner's solution to an appropriate example solution. In several evaluation studies, we examined whether these feedback strategies can help students to improve their solutions. The evaluation confirmed our expectations concerning the proposed feedback provision strategies. *Highlighting* of potentially erroneous parts in students' solutions makes sense according to the experts, particularly for solutions which are assessed (very) poor. Also, students rated the provision of feedback as described in Section 2 as valuable. In a qualitative analysis of student solutions, the positive effect of feedback provision was confirmed.

To automate the selection of suitable examples in a given feedback scenario, we proposed the use of ML techniques which automatically structure a solution space (as defined by a set of student solutions for a task), thereby exploiting clusters of similar solutions. As a major challenge for this approach, we recognised that the proximity measure is of substantial concern to exhibit structure in the solution space and to facilitate the automatic selection. We have discussed three domain-agnostic choices for

proximity measures and provided initial quantitative evidence that clustering based on these measures can adequately capture the structure in real solution sets. In future research, we want to investigate how to automatically adapt a parameterised metric according to the given data, e.g., metrics which can take into account the discriminative relevance of certain parts of a given solution. Therefore, the transfer of *relevance learning* approaches known in the ML literature would offer a promising theoretical foundation, see Schneider et al. (2009) for example.

Since our method uses generic data representations and feedback strategies, it could be integrated easily into existing ITS software. Compared to traditional ITSs, we can summarise several practical advantages of our approach: it is possible to generate automatic feedback in situations where formalised domain knowledge is missing or even impossible, processing and analysing solutions by hand is no longer necessary to adapt an ITS to the learning task or domain, and the approach is applicable to a wide range of ITS domains using the same generic mechanisms. While domain-agnostic automatic feedback provision poses a great challenge, we have introduced principles for the integration in practical modern ITSs.

However, since the proposed approach involves state-of-the-art ML techniques, it currently requires administrative users such as tutors or domain experts to have some background knowledge about the underlying mechanisms. In our future work, we will therefore further advance the implementation of a domain-independent middleware (Gross et al., 2013) which facilitates the construction of intelligently supported learning systems independently from the underlying (formalised) domain knowledge using typical re-usable components of ITSs, exchangeable plug-ins, and ML techniques as described above. This middleware will then enable ITS researchers and developers to use the system to process and analyse student solutions independently from the domain being taught, and to request feedback based on sample solutions.

Currently, our feedback method relies on representative solutions for the different strategies to solve the task. In future research, we will investigate other possibilities by providing feedback based on the closest known correct solution instead of a very limited number of representatives only on the one hand, and will refer to a more detailed proximity measure which is able to compare programs in a more fine-grained way.

Acknowledgements

This work was supported by the German Research Foundation (DFG) under the grant ‘FIT – learning feedback in intelligent tutoring systems’ (PI 767/6 and HA 2719/6). The FIT project is part of the priority programme (SPP) ‘Autonomous learning’.

References

Ahtiainen, A., Surakka, S. and Rahikainen, M. (2006) ‘Plaggie: GNU-licensed source code plagiarism detection engine for Java exercises’, in *Proceedings of the 6th Baltic Sea Conference on Computing Education Research: Koli Calling, Baltic Sea ‘06*, pp.141–142, ACM, New York, USA.

Aleven, V. and Koedinger, K. (2002) ‘An effective metacognitive strategy: learning by doing and explaining with a computer-based cognitive tutor’, *Cognitive Science*, Vol. 26, No. 2, pp.147–179.

Aleven, V., Ashley, K.D., Lynch, C. and Pinkwart, N. (Eds.) (2006) *Proceedings of the Workshop on Intelligent Tutoring Systems for Ill-Defined Domains at the 8th International Conference on Intelligent Tutoring Systems (ITS)*, Jhongli, Taiwan, National Central University.

Aleven, V., Ashley, K.D., Lynch, C. and Pinkwart, N. (Eds.) (2007) *Proceedings of the Workshop on AIED Applications for Ill-Defined Domains at the 13th International Conference on Artificial Intelligence in Education (AIED)*, Los Angeles, CA, USA.

Aleven, V., Ashley, K.D., Lynch, C. and Pinkwart, N. (Eds.) (2008) *Proceedings of the Workshop on Intelligent Tutoring Systems for Ill-Defined Domains at the 9th International Conference on Intelligent Tutoring Systems (ITS)*, Montreal, Canada.

Bailey, M., Lin, K-I. and Sherrell, L. (2012) 'Clustering source code files to predict change propagation during software maintenance', in *Proceedings of the 50th Annual Southeast Regional Conference, ACM-SE '12*, pp.106–111, ACM, New York, NY, USA.

Brusilovsky, P. and Yudelson, M. (2008) 'From webex to navex: interactive access to annotated program examples', *Proceedings of the IEEE*, June, Vol. 96, No. 6, pp.990–999.

Chi, M.T.H., Bassok, M., Lewis, M.W., Reimann, P. and Glaser, R. (1989) 'Self-explanations: how students study and use examples in learning to solve problems', *Cognitive Science*, Vol. 13, No. 2, pp.145–182.

Cho, K. and Schunn, C.D. (2007) 'Scaffolded writing and rewriting in the discipline: a web-based reciprocal peer review system', *Computers & Education*, April, Vol. 48, pp.409–426.

Douglas, R. and Sejnowski, T. (2007) *Report of the NSF Workshop on Future Challenges for the Science and Engineering of Learning*, July.

Fournier-Viger, P., Nkambou, R., Nguifo, E.M. and Mayers, A. (2010) 'Its in ill-defined domains: toward hybrid approaches', in *ITS 2010*, pp.749–751, Springer.

Goodlad, S. and Hirst, B. (1989) *Peer Tutoring: A Guide to Learning by Teaching*, Kogan Page.

Gross, S., Mokbel, B., Hammer, B. and Pinkwart, N. (2013) 'Towards a domain-independent its middleware architecture', in N-S. Chen, R. Huang, Kinshuh, Y. Li and D.G. Sampson (Eds.): *Proceedings of the 13th IEEE International Conference on Advanced Learning Technologies (ICALT)*, pp.408–409.

Gusfield, D. (1997) *Algorithms on Strings, Trees, and Sequences – Computer Science and Computational Biology*, Cambridge University Press.

Hammer, B. and Hasenfuss, A. (2007) 'Relational neural gas', in J. Hertzberg, M. Beetz and R. Englert (Eds.): *KI 2007: Advances in Artificial Intelligence, 30th Annual German Conference on AI, KI 2007*, Vol. 4667 of *Lecture Notes in Artificial Intelligence*, pp.190–204, Springer, Berlin.

Hammer, B. and Jain, B.J. (2004) 'Neural methods for non-standard data', in *ESANN*, pp.281–292.

Hammer, B. (2002) 'Compositionality in neural systems', in M. Arbib (Ed.): *Handbook of Brain Theory and Neural Networks*, 2nd ed., pp.244–248, MIT Press.

Hasenfuss, A. and Hammer, B. (2010) 'Topographic mapping of large dissimilarity datasets', *Neural Computation*, Vol. 22, No. 9, pp.2229–2284.

Jonassen, D.H. (1997) 'Instructional design models for well-structured and ill-structured problem-solving learning outcomes', *Educational Technology Research and Development*, Vol. 45, No. 1, pp.65–94.

Kohonen, T. (1995) *Self-organizing Maps*, Springer.

Kop, R. (2011) 'The challenges to connectivist learning on open online networks: learning experiences during a massive open online course', *The International Review of Research in Open and Distance Learning, Special Issue-Connectivism: Design and Delivery of Social Networked Learning*, Vol. 12, No. 3.

Kuhn, A., Ducasse, S. and Gírba, T. (2007) 'Semantic clustering: identifying topics in source code', *Information and Software Technology*, *12th Working Conference on Reverse Engineering*, Vol. 49, No. 3, pp.230–243.

Kulik, J.A. and Kulik, C. (1988) 'Timing of feedback and verbal learning', *Rev. of Educational Research*, Vol. 58, No. 1, pp.79–97.

Li, M., Chen, X., Li, X., Ma, B. and Vitanyi, P. (2004) 'The similarity metric', *Information Theory, IEEE Transactions on*, December, Vol. 50, No. 12, pp.3250–3264.

Liu, Y., Li, Z., Xiong, H., Gao, X. and Wu, J. (2010) 'Understanding of internal clustering validation measures', in G.I. Webb, B. Liu, C. Zhang, D. Gunopulos and X. Wu (Eds.): *ICDM*, pp.911–916, IEEE Computer Society.

Loll, F. and Pinkwart, N. (2013) 'Lasad: flexible representations for computer-based collaborative argumentation', *International Journal of Human-Computer Studies*, Vol. 71, No. 1, pp.91–109.

Lynch, C., Ashley, K.D., Aleven, V. and Pinkwart, N. (2006) 'Defining ill-defined domains: a literature survey', in V. Aleven, K.D. Ashley, C. Lynch and N. Pinkwart (Eds.): *Proceedings of the Workshop on Intelligent Tutoring Systems for Ill-Defined Domains at the 8th International Conference on Intelligent Tutoring Systems (ITS)*, pp.1–10, National Central University, Jhongli, Taiwan.

Lynch, C., Ashley, K.D., Mitrovic, A., Dimitrova, V., Pinkwart, N. and Aleven, V. (Eds.) (2010a) *Proceedings of the Workshop on Intelligent Tutoring Systems for Ill-Defined Domains at the 10th International Conference on Intelligent Tutoring Systems (ITS)*, Pittsburgh, PA, USA.

Lynch, C., Ashley, K.D., Pinkwart, N. and Aleven, V. (2010b) 'Concepts, structures, and goals: redefining ill-definedness', *International Journal of Artificial Intelligence in Education*, Vol. 19, No. 3, pp.253–266.

Müller, M. (2007) *Information Retrieval for Music and Motion*, Springer-Verlag New York, Inc., Secaucus, NJ, USA.

Melis, E. (2005) 'Choice of feedback strategies', in *Cognition and Exploratory Learning in the Digital Age (CELDA 2005), IADIS*, Vol. 12.

Merrill, D.C., Reiser, B.J., Ranney, M. and Trafton, J.G. (1992) 'Effective tutoring techniques: a comparison of human tutors and intelligent tutoring systems', *Journal of the Learning Sciences*, Vol. 2, No. 3, pp.277–305.

Mitrovic, A., Mayo, M., Suraweera, P. and Martin, B. (2001) 'Constraint-based tutors: a success story', in *Proceedings of the 14th International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems*, pp.931–940, Springer-Verlag, London, UK.

Mitrovic, A., Ohlsson, S. and Barrow, D.K. (2013) 'The effect of positive feedback in a constraint-based intelligent tutoring system', *Computers & Education*, Vol. 60, No. 1, pp.264–272.

Mory, E.H. (2004) *Feedback Research Revisited*, pp.745–783, Association for Educational Communications and Technology.

Moschitti, A., Pighin, D. and Basili, R. (2008) 'Tree kernels for semantic role labeling', *Comput. Linguist.*, June, Vol. 34, No. 2, pp.193–224.

Murray, T., Blessing, S. and Ainsworth, S. (Eds.) (2003) *Authoring Tools for Advanced Technology Learning Environments*, Kluwer Academic Publishers, Dordrecht.

Neuhaus, M. and Bunke, H. (2006) 'Edit distance-based kernel functions for structural pattern classification', *Pattern Recognition*, Vol. 39, No. 10, pp.1852–1863.

Nkambou, R., Fournier-Viger, P. and Nguifo, E.M. (2011) 'Learning task models in ill-defined domain using an hybrid knowledge discovery framework', *Know-Based Syst.*, February, Vol. 24, No. 1, pp.176–185.

Ogan, A., Aleven, V. and Jones, C. (2009) 'Advancing development of intercultural competence through supporting predictions in narrative video', *International Journal of Artificial Intelligent in Education*, August, Vol. 19, No. 3, pp.267–288.

Ohlson, S. (1996) 'Learning from performance errors', *Psychological Review*, Vol. 103, pp.241–262.

Pekalska, E. and Duin, R.P. (2005) *The Dissimilarity Representation for Pattern Recognition. Foundations and Applications*, World Scientific.

Piaget, J. (1972) *The Psychology of the Child*, Basic Books, New Yor, NY.

Pople, H.E. (1982) 'Heuristic methods for imposing structure on ill-structured problems: the structuring of medical diagnostics', in P. Szolovits (Ed.): *AI in Medicine*, pp.119–190, Westview Press, Boulder (CO).

Reitman, W.R. (1965) *Cognition and Thought: An Information Processing Approach*, John Wiley and Sons, New York, NY.

Renkl, A., Stark, R., Gruber, H. and Mandl, H. (1998) 'Learning from worked-out examples: the effects of example variability and elicited self-explanations', *Contemporary Educational Psychology*, Vol. 23, No. 1, pp.90–108.

Roll, I., Aleven, V., McLaren, B.M. and Koedinger, K.R. (2011) 'Improving students' help-seeking skills using metacognitive feedback in an intelligent tutoring system', *Learning and Instruction*, Vol. 21, No. 2, pp.267–280.

Salton, G. and Buckley, C. (1988) 'Term-weighting approaches in automatic text retrieval', *Inf. Process. Manage.*, August, Vol. 24, No. 5, pp.513–523.

Schneider, P., Biehl, M. and Hammer, B. (2009) 'Adaptive relevance matrices in learning vector quantization', *Neural Comput.*, December, Vol. 21, No. 12, pp.3532–3561.

Simon, H.A. (1973) 'The structure of ill-structured problems', *Artificial Intelligence*, Vol. 4, No. 3, pp.181–201.

Striewe, M. and Goedicke, M. (2011)'Using run time traces in automated programming tutoring', in G. Rößling, T.L. Naps and C. Spannagel (Eds.): *ITiCSE*, pp.303–307, ACM.

Striewe, M. and Goedicke, M. (2013) 'Analyse von programmieraufgaben durch softwareproduktmetriken', in A. Spillner and H. Lichter (Eds.): *SEUH*, Vol. 956 of *CEUR Workshop Proceedings*, pp.59–68, CEUR-WS.org.

Topping, K. (1998) 'Peer assessment between students in colleges and universities', *Rev. of Educational Research*, Vol. 68, No. 3, pp.249–276.

Tsovaltzi, D., Melis, E., McLaren, B., Meyer, A-K., Dietrich, M. and Goguadze, G. (2010) 'Learning from erroneous examples: when and how do students benefit from them?', in M. Wolpers, P. Kirschner, M. Scheffel, S. Lindstaedt and V. Dimitrova (Eds.): *Sustaining TEL: From Innovation to Learning and Practice*, Vol. 6383 of *Lecture Notes in Computer Science*, pp.357–373, Springer, Berlin Heidelberg.

Ullmann, J.R. (1976) 'An algorithm for subgraph isomorphism', *J. ACM*, January, Vol. 23, No. 1, pp.31–42.

van der Maaten, L. and Hinton, G. (2008) 'Visualizing data using t-sne', *J. of Mach. Learn. Res.*, November, Vol. 9, pp.2579–2605.

Verleysen, M. and François, D. (2005) 'The curse of dimensionality in data mining and time series prediction', in *Computational Intelligence and Bioinspired Systems*, pp.758–770, Springer.

Voss, J.F. (2006) 'Toulmin's model and the solving of ill-structured problems', in D.H.B. Verheij (Ed.): *Arguing on the Toulmin Model: New Essays in Argument Analysis and Evaluation*, pp.303–311, Springer, Berlin.

Walker, E., Ogan, A., Aleven, V. and Jones, C. (2008) 'Two approaches for providing adaptive support for discussion in an ill-defined domain', in *Proc. of a Workshop at ITS*, Montreal Canada, June 23, pp.1–12.

Walker, E. , Rummel, N., Walker, S. and Koedinger, K. (2012) 'Noticing relevant feedback improves learning in an intelligent tutoring system for peer tutoring', in S. Cerri, W. Clancey, G. Papadourakis and K. Panourgia (Eds.): *Intelligent Tutoring Systems*, volume 7315 of *Lecture Notes in Computer Science*, pp.222–232, Springer, Berlin Heidelberg.

Wittwer, J. and Renkl, A. (2010) 'How effective are instructional explanations in example-based learning? A meta-analytic review', *Educational Psychology Review*, Vol. 22, No. 4, pp.393–409.

Zakharov, K., Mitrovic, A. and Ohlsson, S. (2005) 'Feedback micro-engineering in eer-tutor', in *Proc. AIED 2005*, pp.718–725, Press.