The Beetle and BeeDiff Tutoring Systems

Charles Callaway, Myroslava Dzikovska, Elaine Farrow,
Manuel Marques-Pita, Colin Matheson, Johanna Moore

Human Communication Research Centre
University of Edinburgh, Edinburgh, United Kingdom
{ccallawa|mdzikovs|efarrow|mmpita|colin|jmoore}@inf.ed.ac.uk

Abstract

We describe two tutorial dialogue systems that adapt techniques from task-oriented dialogue systems to tutorial dialogue. Both systems employ the same reusable deep natural language understanding and generation components to interpret students' written utterances and to automatically generate adaptive tutorial responses, with separate domain reasoners to provide the necessary knowledge about the correctness of student answers and hinting strategies. We focus on integrating the domain-independent language processing components with domain-specific reasoning and tutorial components in order to improve the dialogue interaction, and present a preliminary analysis of BeeDiff's evaluation.

Index Terms: tutoring systems, dialogue, deep processing

1. Introduction

One-on-one tutoring is known to be better in helping students learn when compared with reading textbooks. Human tutors typically produce effect sizes of up to two standard deviations compared to unsupervised reading [1]. Tutorial systems, in particular cognitive tutors, improve over reading alone but still result in only up to 1 standard deviation effect size [2]. One potential reason for this difference is more interactive dialogue, which allows students to freely ask questions and tutors to adapt their direct feedback and presentation style to an individual student's needs. The specific properties of dialogue that help students learn are still under study, but possibilities include allowing students to explain their actions [3], adapting tutorial feedback to the learner's level, affective factors [4], and the interactive nature of human-human dialogue.

Some tutorial dialogue systems use NLP techniques to analyze student responses to "why" questions [5,6,7]. However, for remediation they revert to scripted dialogue, with short-answer questions and canned feedback. The resulting dialogue may be redundant in ways detrimental to student understanding [8] and allows for only limited adaptivity [9].

In contrast, cognitive tutors build up a detailed model of a student's knowledge, and provide hints and feedback tailored to that model [10]. They can also implement more complex tutoring strategies by decomposing problems and asking students questions, as in the Ms Lindquist tutor [11]. But since dialogue management is tightly bound to a domain model, these systems are difficult to design and port to new domains.

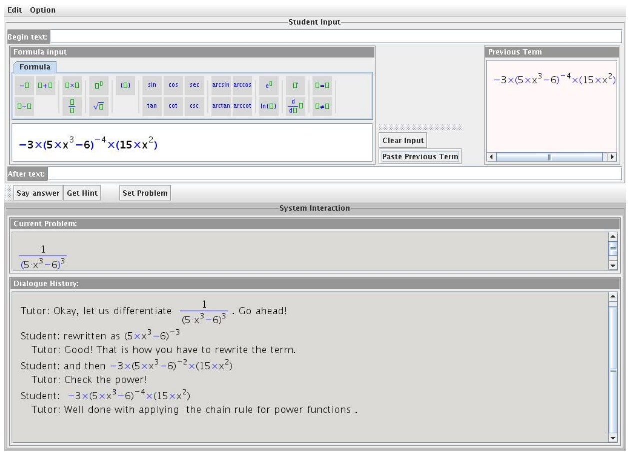
The long-term goal of our research is to build tutorial dialogue systems that approximate the properties of human-human dialogue to improve student learning and engagement. As a first step, we adapted the techniques from task-oriented dialogue systems to tutorial dialogue to support more flexible and adaptive dialogue. We describe two tutorial dialogue systems built using the Information State Update approach for dialogue management and generic components for deep natural language understanding and generation. Typed student utterances are parsed and interpreted, taking dialogue context into account, and tutorial feedback is generated adaptively based on the student model.

2. The BeeDiff Tutor

The **BeeDiff** tutor helps students solve symbolic differentiation problems, assuming that students are learning lesson content in lectures. During each tutoring session, the student is presented with a series of differentiation problems to solve. Students can enter answer solutions immediately, or work incrementally on problems by doing individual steps. They can also request help and ask simple questions such as "Should I apply the chain rule?" and "What is the product rule?"

Student input is relatively limited and often consists only of mathematical formulas (Fig. 1), but **BeeDiff** generates adaptive feedback based on the diagnoser output (expert and buggy rules applied by the student), a notion of student performance and the dialogue history. The types of feedback available and methods for selecting between them are informed by a corpus of 19 human-human tutorial interactions [12].

As an example, if the question is to differentiate *x^-3* and the student answers incorrectly with *-3x^-2*, a high performing student will get "*Not quite"* as feedback, a good student "*Check the power*", a fair student "*Check the power, it shouldn't be -3*", and the system might bottom out for poor students, saying "*Not quite. The answer should be -3*x^-4*". These different adaptive feedback messages are generated automatically from a single



$-3\times(5\times x^3-6)^{-4}\times(15\times x^2)$

$-3\times(5\times x^3-6)^{-4}\times(15\times x^2)$

$\frac{1}{(5\cdot x^3-6)^3}$

Tutor: Okay, let us differentiate $\frac{1}{(5\cdot x^3-6)^3}$. Go ahead!
Student: rewritten as $(5\times x^3-6)^{-3}$
Tutor: Good! That is how you have to rewrite the term.
Student: and then $-3\times(5\times x^3-6)^{-2}\times(15\times x^2)$
Tutor: Check the power!
Student: $-3\times(5\times x^3-6)^{-4}\times(15\times x^2)$
Tutor: Well done with applying the chain rule for power functions .

Figure 1: Screenshot of the BeeDiff tutor

diagnosis data structure (Section 5). In addition, the feedback can be adjusted based on the student confidence (Section 4.1).

3. The Beetle Tutor

The **Beetle** tutor is designed to teach students basic electricity and electronics concepts. It is built around a pre-planned course where the students alternate reading with exercises involving answering "*Why*" questions and interacting with a circuit simulator. Unlike **BeeDiff**, here students are primarily focused on learning concepts such as voltage and current and interpreting circuit behavior, rather than repeated problem solving (Fig. 2). System development was informed by a corpus of 36 dialogues collected with 3 different tutors, each covering a student and tutor completing a series of 3 lessons. Initial system design was based on post-lesson discussion with tutors, but we are currently analyzing the data to determine those dialogue features that correlate with learning gain in order to improve the system.

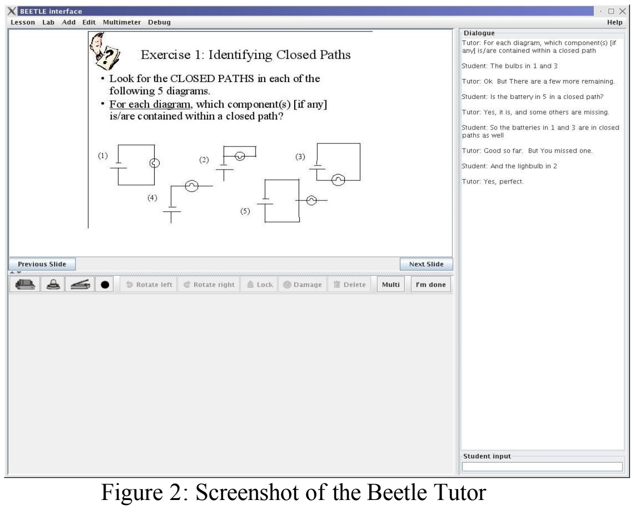
Since this is a conceptual domain, for most exercises there is no structured sequence of steps that the students should follow. The only requirement is to name a correct set of objects and relationships in their response. We model the process of building an answer to an exercise as co-constructing a solution, where the student and tutor may each contribute parts. For instance, consider the identification-oriented question "*For each circuit, which components are in a closed path*?" The solution can be built gradually, with the student naming different components, and the system providing feedback until the list is complete.

We also apply this generic process of gradually building up a solution to giving explanations. For example, for the question "*What is required for a light bulb to light?*" the student may reply "*The bulb must be in a closed path*", which is correct but incomplete. The system might then ask "*Correct, but is that everything?*" to prompt the student to also mention the battery.

We are currently developing a diagnoser and tutorial module to provide flexible and adaptive remediation strategies for cases when students make mistakes, discussed in Section 5.

4. Architecture

Both systems are built using a generic architecture with the **TRINDIKIT** dialogue manager [13]. The dialogue manager is invokes the appropriate interpretation and generation components, maintains the dialogue state, and implements turn taking and other generic dialogue behaviors. Parsing, reference resolution and surface generation are implemented using generic components shared between both **Beetle** and **BeeDiff**.



- Look for the CLOSED PATHS in each of the following 5 diagrams.
- For each diagram, which component(s) [if any] is/are contained within a closed path?

Tutor: For each diagram, which component(s) [if any] is/are contained within a closed path

Student: The bulbs in 1 and 3

Tutor: Ok But There are a few more remaining.

Student: Is the battery in 5 in a closed path?

Tutor: Yes, it is, and some others are missing.

Student: So the batteries in 1 and 3 are in closed paths as well

Tutor: Good so far. But You missed one.

Student: And the lighbulb in 2

Tutor: Yes, perfect.

Figure 2: Screenshot of the Beetle Tutor

4.1. Interpretation

Our interpretation module uses a 2-layer architecture for interpretation supported by the TRIPS dialogue parser [14]. Utterance meaning is represented using a domain-independent semantic ontology and syntax, which is connected to the domain-specific knowledge representation by mapping between the domain-independent and domain-specific ontologies.

The domain-independent linguistic features output by the parser are used to help reference resolution, detect hedged answers by using the features described in [15], and to detect indirect speech acts. For example, utterances that are statements on the surface, such as "I need help", are reinterpreted as requests for help based on the linguistic features. The utterance content is then mapped to a domain-specific representation.

In the **BeeDiff** domain, the domain-specific content is used to query the domain reasoner about the rule applications. In **Beetle**, the domain content is used to query the knowledge base and to check student answers.

Consider the student answer to the following problem in the **Beetle** domain:

Problem: For each circuit, which components are in a closed path?

Student: Are the bulbs in 1 and 3 in closed paths?

First, the system resolves "*the bulbs in 1 and 3*" to corresponding object IDs in the knowledge base, for example, `LB-13-1-1` and `LB-13-3-1`, and query the database to determine if these bulbs are indeed in closed path. Next, a set of relationships is extracted from the sentence, in this case `(path ?p) (is-closed ?p t) (bulb LB-13-1-1) (in LB-13-1-1 ?p)` This interpretation is passed to the dialogue manager, which attempts to determine if the statement is a possible answer to the question. It computes an intersection between the concepts mentioned in the question (components, closed paths) and the concepts mentioned in the answer. Since there is a good match, the dialogue manager can interpret the statement as an answer, verify its correctness and remediate appropriately. In contrast, if the student said "*Is the bulb in 1 connected to the battery*" in this context, this statement would be interpreted as a question and answered by querying the domain knowledge base.

4.2. Dialogue Management

The context maintenance and dialogue strategies for dealing with questions and answers, clarifications and uninterpretable input are not dependent on the domain. They are separated from the tutorial and domain reasoning, allowing us to share portions of dialogue management between the systems. Section 5 discusses the domain-specific diagnosis and tutorial reasoning, and their interaction with dialogue management.

The dialogue manager (DM) uses the dialogue context to interpret student dialogue acts. In particular, hedged answers phrased as check questions described above are common in both of our domains. They are intepreted by a generic 3-step process: the DM calls an interpretation module to get the sentence mood and a domain-specific representation of an utterance; the DM uses a domain-specific module to decide if the semantic content of an utterance is relevant to the current question; finally, the DM either selects a dialogue strategy to address the student turn as an answer to the problem, or treats it as a student question.

The DM also maintains the dialogue and diagnosis histories, recording all utterances made by the system and the student, and all student mistakes. These are used to produce more coherent dialogue with appropriate discourse cues. For example, when a student repeats the same mistake in the **BeeDiff** domain, the feedback may be "*You've correctly differentiated the inner layer, but you're still missing the minus sign.*" The two clauses are joined by a contrast relationship, and the second indicates that an error was repeated by using the adverbial "*still*".

4.3. Generation

Our current implementation uses a custom utterance generation component and the **StoryBook** [16] deep text generator modified to work in a dialogue context. StoryBook automatically handles low-level linguistic details such as inserting pronouns, modals, modifiers, and aggregating small dialogue acts with discourse markers into fluent tutor turns. While StoryBook itself is domain-independent, a domain-aware component is needed to convert tutorial plans produced by the dialogue manager (representing sets of dialogue acts) into the speech-act-based input expected by StoryBook. The Diag-NLP system [18] has also attempted to integrate deep generation in a tutoring system, but it does not engage the student in dialogue.

Deep linguistic representations are necessary for longer and more sophisticated tutorial utterances because they are typically composed of a series of smaller dialogue acts which must be linked together with discourse markers like "because", "then" and "where" or via other syntactic constructions like relative clauses. If dialogue acts are represented shallowly (e.g., as text strings), it becomes prohibitively expensive to encode all combinatorially possible sequences of dialogue acts. In the **BeeDiff** domain, the generation component can realize the following set of three dialogue acts as a single fluent sentence:

```
((context {confidence} {performance} {difficulty})
  (id1 (join id2 id3 id4))
  (id2 (accept incorrect none))
  (id3 (reassert location power incorrect))
  (id4 (assert location power bad_value "-2")))
```

"Not really, you've still got the power wrong, it shouldn't be -2"

We use standard relations from Rhetorical Structure Theory (e.g., *elaboration*, *sequence* or *consequence*) to connect dialogue acts in a meaningful way. Our dialogue acts are represented with flat semantics and include explicit contextual information that allows us to vary realization appropriately.

5. Diagnosis and Tutorial Reasoning

Diagnosis of student input in both domains needs to provide a detailed analysis of the output which could be used as a basis for adaptive generation. However, the specific implementation is different depending on the domain.

In the **BeeDiff** domain, each diagnosis consists of the following parts: a rule code, a rule type (expert or buggy), rule name, wrong part of the answer, how it would be if corrected, the rule input, and the rule output. For example, if the problem is x^{-3} and the student enters $-3*x^{-2}$, the diagnoser will return:

```
[bg-pl111,type(buggy),rule(power_rule),
 expr(x^(-3)),student_answer(-3*x^(-2)),
 wrong_part(exponent,-2),
 corrected_part(exponent,-4)]
```

This data structure can then be used to generate feedback at a different levels of specificity by selecting individual structural elements to present to the student (we also tailor hint specificity with factors such as confidence and performance). We currently implement a simple strategy with 4 levels of specificity: contentless (e.g., "Almost"), general error location (only the label of the wrong part is used to describe the problem), specific error location (both the label and the expression for the wrong part is used to describe a problem) and bottom-out (the whole answer is given to the student) [17]. Many other strategies are possible, for example partial bottom outs, where the system gives the student the correct value for the part, but not the whole answer. These can be easily changed in the tutorial module to experiment with different options for adapting feedback. Because the tutorial module takes generic data structures as input, a large system is easier to adjust than prior methods that pre-author specific messages attached to individual rules.

The current remediation strategy in the **BeeDiff** domain is to focus on the specific problems in the student's answer identified with buggy rules, namely to point out the location of the mistake at different levels of specificity, which is the strategy most frequently used by human tutors [17]. In **Beetle**, even though common student mistakes and misconceptions occur, our data collection and discussions with tutors indicate that the tutors don't specifically address misconceptions in their remediations unless the misconception is stated explicitly in words. In addition, unlike the **BeeDiff** domain, **Beetle** does not have a problem-solving model with a specified number and order of steps for each problem.

Therefore, in the **Beetle** domain we represent student answers as lists of objects and relationships expressed in a logical form language. Unless the student input matches a specific misconception, the diagnoser tries to classify each object and relationship that the student mentioned in one of the 5 classes: correct parts - objects and relationships present in the standard answer; direct errors - objects and relationships directly contradicting the standard answer (*e.g.*, the student says the terminals are separated when they are connected); irrelevant parts - objects and relationships not present in the standard answer, perhaps because an irrelevant (and therefore incorrect) reason was given in the explanation, or because the student was too verbose; and missing parts – those present in the standard answer but not present in the student's answer.

We are currently developing a tutorial component that can use this above encoding to address reasoning errors and incomplete answers without the need to tie remediations to specific buggy rules. The system will first attempt to address direct errors, then missing parts. If there are no correct parts or direct errors, the system will assume the student selected an irrelevant reason and address the irrelevant parts of the answer if possible. If the answer is fully correct, and the irrelevant relationships are correct (the student was too verbose), the tutor will not address these, but may apply a "reiterate key point" reinforcement strategy by restating the answer in a briefer form.

We implemented a generic dialogue strategy that allows the system to gradually accumulate a student answer over multiple turns if the original answer was incomplete, by maintaining and updating a cumulative answer state. We still need to address the issue of how to remediate those student answers that directly contradict the tutor's statements. One common strategy we observed in the Beetle domain is breaking the problem into smaller steps, somewhat similar to what is done in the Ms Lindquist tutor. We are planning to develop parametrization of problems that would allow for such a strategy in this domain, since our architecture can provide dialogue support for more flexible and adaptive dialogue.

6. Evaluation

We have just completed an evaluation of learning gain on the **BeeDiff** tutoring system against a similar non-dialogue, finite-state system without adaptive feedback in the same domain (the analysis presented here is both preliminary and abridged). We ran 28 undergraduate students, discarding subjects with pre-test scores greater than 60% or less than 10%. 15 were assigned to the non-dialogue condition, and 13 to the adaptive dialogue condition The subjects solved a pre-test, spent one hour working with the system, and completed the post-test.

The average pre-test score for the non-dialogue condition was 30 (σ=14.63), and for the adaptive dialogue condition 26.15 (σ=8.69). The distributions of pre-test scores did not differ between conditions (2-tailed t-test, p=0.42, F(26)=0.83). The average post-test scores were 49.74 (σ=35.82) and 75.14 (σ=26.48) respectively. Students learned in both conditions: the pre-test distribution was significantly different from post-test for both (2-tailed paired t-test; cond1: p=0.02, F(14)=2.49; cond2: p<0.01, F(12)=8.25). The resulting learning gains were 0.31 (σ=0.49) for the first condition, and 0.68 (σ=0.31) for the adaptive dialogue condition.

To verify that the learning gain was significantly different between conditions, we ran a one-way analysis of covariance (ANCOVA) using the pre-test as a covariate, system condition as the independent variable, and post-test as the dependent variable. The adjusted mean post-test scores were 46.87 for the first condition, and 78.47 for the dialogue condition. The difference is statistically significant, F(1,25) = 10.43, p=0.003.

7. Discussion and Conclusions

The evaluation we presented shows that in the **BeeDiff** domain, the adaptive system outperformed the hand-authored non-adaptive system. We believe that this is due, in part, to the adaptivity and to the more detailed hints and feedback possible with generic dialogue management. To verify this hypothesis, we are planning another evaluation, where we would compare learning gain with two versions of our system: one with adaptive feedback, and one where the same feedback (specific error location) is given to all mistakes. We expect that the results would be similar to the evaluation we just conducted, but this would allow for the maximally controlled comparison.

Our systems rely on diagnosers that produce detailed analyses of correct and incorrect parts of student answers. This is a more complex model of the dialogue management than is used, for example, in Why2-Atlas [6] or AUTOTUTOR [7] systems. These systems attempt to identify incorrect parts of the student answer as stemming from a specific misconception (e.g. impetus misconception), and use scripted dialogues dependent only on the misconception to remediate. This allows the system designers to implement dialogues specific to individual misconceptions, but at the expense of adaptivity to student model, since authoring individual dialogues for every possible combination of contextual factors is very labor intensive.

Our approach requires implementing an additional tutoring module, but it supports adaptivity by taking into account the richer diagnosis representation, as well as dialogue context. In that respect it is more similar to the approaches used in cognitive tutors, such as Ms Lindquist [11] or the PACT geometry tutor [5], which tailor their messages to the specific student input. However, the use of dialogue management techniques developed in the task-oriented dialogue community allows us to provide for more flexible, mixed initiative dialogue where the students can ask questions, and the system can also use features of natural language input to detect and react to student affect, such as the level of student confidence.

Acknowledgements: This work was supported under the 6th Framework Programme of the European Commission, Ref. IST-507826, and by a grant from The Office of Naval Research N000149910165. We thank Gwen Campbel, Natalie Martin, Leslie Butler, Kaska Porayska-Pomsta for help with data collection and system design.

8. References

[1] Bloom, B.S. "The 2 Sigma Problem: The search for methods of group instruction as effective as one-to-one tutoring", Educational Researcher, 13:4-16, 1984.

[2] Anderson, J. R., Corbett, A. T., Koedinger, K. R., and Pelletier, R. "Cognitive Tutors: Lessons Learned", The Journal of the Learning Sciences, 4(2):167-207, 1995.

[3] Chi, M. T. H., Bassok, M., Lewis, M. W., Reimann, P., and Glaser, R. "Self-Explanations: How Students Study and Use Examples in Learning to Solve Problems", Cognitive Science, 13(2):145-182, 1989.

[4] Litman, D. J., and Forbes-Riley, K. "Predicting Student Emotions in Computer-Human Tutoring Dialogues", in Proceedings of ACL'04, 2004.

[5] Aleven, V., Popescu, O., and Koedinger, K. R., "Towards tutorial dialog to support self-explanation: Adding natural language understanding to a cognitive tutor", in Proceedings of AI in Education, 2001.

[6] Jordan, P., Makatchev, M., Pappuswamy, U., VanLehn, K., and Albacete, P., "A Natural Language Tutorial Dialogue System for Physics", in Proceedings of FLAIRS'06, 2006.

[7] Graesser, A. C., Wiemer-Hastings, P., Wiemer-Hastings, P., and Kreuz, R., "AutoTutor: A simulation of a human tutor", Cognitive Systems Research, 1:35-51, 1999.

[8] Jordan, P., Albacete, P., and VanLehn, K., "Taking Control of Redundancy in Scripted Tutorial Dialogue", Proceedings of AIED2005, pages 314-321, 2005.

[9] Jordan, P. "Using Student Explanations as Models for Adapting Tutorial Dialogue", FLAIRS Conference, 2004.

[10] Anderson, J. R., Boyle, D. F., & Reiser, B. J. "Intelligent tutoring systems". Science, 228, 456-462, 1985.

[11] Heffernan, N. T., & Koedinger, K. R. "An Intelligent Tutoring System Incorporating a Model of an Experienced Human Tutor" In Proceedings of ITS-02, 2002.

[12] Dzikovska, M.O., Reitter, D., Moore, J.D. and Zinn, C. "Data-driven Modeling of Human Tutoring in Calculus". In Proceedings. of the ECAI-06 Workshop on Language-enhanced Educational Technology, 2006.

[13] Larsson, S. and Traum, D. R., "Information state and dialogue management in the TRINDI dialogue move engine toolkit", Natural Language Engineering, 6(3-4):323-340, 2000.

[14] Allen, J., Dzikovska, M., Manshadi, M., and Swift, M. "Deep Linguistic Processing for Spoken Dialogue Systems", In proceedings of EACL-07 workshop on Deep Linguistic Processing, 2007.

[15] Bhatt, K., Evens, M., and Argamon, S., "Hedged Responses and Expressions of Affect in Human/Human and Human/Computer Tutorial Interactions", in Proceedings of COGSCI, 2004

[16] Callaway, C. B., and Lester, J. C., "Narrative Prose Generation", Artificial Intelligence, 139(2):213-252, 2002.

[17] Callaway, C. B. and Moore, J. D. "Determining tutorial remediation strategies from a corpus of human-human tutoring dialogues", in Proc. of ENLG, 2007.

[18] Di Eugenio, B., Fossati, D., Yu, D., Haller, S., and Glass, M., "Natural Language Generation for Intelligent Tutoring Systems: A case study", in Proc. of AIED, 2005.