Automating Hinting in Mathematical Tutorial Dialogue

Armin Fiedler
Department of Computer Science
Saarland University
afiedler@cs.uni-sb.de

Dimitra Tsovaltzi
Department of Computational Linguistics
Saarland University
dimitra@coli.uni-sb.de

Thematic Session: Adaptation and learning in spoken dialogue systems

Abstract

In spite of psychological substantiation of hinting, most intelligent tutoring systems do not systematically produce hints. In this paper we present a taxonomy of hints for tutoring mathematics. Based on this taxonomy, we suggest an algorithm for the production of hints.

1 Introduction

Although there has been psychological evidence (Chi et al., 1994; Rosé et al., 2001) for the high educational effect of hinting, most intelligent tutoring systems do not take advantage of the findings. Only little systematic research in automating hinting has been done (Hume et al., 1996b; Fiedler and Horacek, 2001; Tsovaltzi, 2001).

In the DIALOG project, we undertake the goal of tackling some problems of mathematical tutoring dialogue systems in the domain of naive set theory (Pinkal et al., 2001). More specifically, DIALOG aims at providing a mathematical knowledge base that would serve as the basis for the study material, a user-adaptive dialogue system modelling tutorial dialogues and an advanced theorem prover fitting the needs of planning for tutorials. In this framework, we have been investigating the automation of the hinting process.

A mathematical tutoring system must be able to tutor proofs in a way that not only helps the student understand the current proof, but also allows for a high learning effect. What is meant by the latter is the ability of the students not only to better understand the problem at hand, but also to generalise and apply the taught strategies on their own later on. We propose to establish those tutoring aims by making use of the *socratic* tutoring method (Person et al., 2000; Rosé et al., 2001). The decisive characteristic of the socratic method is the use of hints in order to achieve self-explanation, as opposed to answers and explanations being provided constantly by the tutor, which is characteristic of the *didactic* method (Rosé et al., 2001; Tsovaltzi and Matheson, 2002).

Hinting is a method that aims at encouraging active learning. It can take the form of eliciting information that the student is unable to access without the aid of prompts, or information which he can access but whose relevance he is unaware of with respect to the problem at hand. Alternatively, a hint can point to an inference that the student is expected to make based on knowledge available to him, which helps the general reasoning needed to deal with a problem (Hume et al., 1996b).

In this paper, we shall first give in Section 2 a taxonomy of hints that we developed for the domain of mathematics tutoring. Next, in Section 3, we shall propose an algorithm for producing hints based on that taxonomy. This hinting algorithm is envisaged as part of a dialogue manager, which accounts also for other types of dialogues, such as clarification sub-dialogues and question answering. Then, in Section 4, the algorithm will be supplemented with an example, before we conclude the paper.

2 A Taxonomy of Hints

In this section we shall explain the philosophy and the structure of our hint taxonomy. We shall also look into the most representative hints that are used in the algorithm. The names of the categories are intended to be as descriptive of the con-

tent as possible, and should in some cases be self-explanatory. Our taxonomy includes more than the hint categories mentioned in this section. The full taxonomy is given in Table 1. Some of the strategies are not real hints (e.g., *point-to-lesson*), but they have been included in the taxonomy because they are part of the general hinting process.

2.1 Philosophy and Structure

Our hint taxonomy was derived with regard to the underlying function that can be common for different surface realisations. The underlying function is mainly responsible for the educational effect of hints. The surface structure, which undoubtedly plays its own significant role in teaching, is yet to be examined in the naive set theory domain.

We defined our hint categories based on the needs in the domain. In order to estimate those needs we defined the relations between the objects in the domain. More specifically, we defined the inter-relations between mathematical concepts as well as between concepts and inference rules, which are used in proving. An additional guide for deriving hint categories that are useful for tutoring in our domain was a previous hint taxonomy, which was derived from the BE&E corpus (Tsovaltzi, 2001).

The structure of the hint taxonomy reflects the function of the hints with respect to the information that the hint addresses or is meant to trigger. In order to capture different functions of a hint we define hint categories across two dimensions.

Before we introduce the dimensions, let us clarify some terminology. In the following, we distinguish performable steps from meta-reasoning. *Performable steps* are the steps that can be found in the formal proof. These include premises, conclusion and inference methods such as lemmata, theorems, definitions of concepts, or calculus level rules (e.g., proof by contradiction). *Meta-reasoning steps* consist of everything that leads to the performable step, but cannot be found in the formal proof. To be more specific, meta-reasoning consists of everything that could potentially be applied to any particular proof. It involves general proving techniques. As soon as a general technique is instantiated for the particular proof, it belongs to the performable step level.

The two hint dimensions consist of the following classes:

1. active vs. passive hints
2. domain-relation vs. domain-object vs. inference-rule vs. substitution vs. meta-reasoning vs. performable-step hints

In the second dimension, we ordered the classes with respect to their subordination relation. We say, that a class is *subordinate* to another one if it reveals more information. In addition, the class of *pragmatic* hints belongs to the second dimension as well, but we define it so that it is not subordinate to any other class and no other class is subordinate to it.

2.2 First Dimension

The first dimension distinguishes between the active and passive function of hints. The difference lies in the way the information to which the tutor wants to refer is approached. The idea behind this distinction resembles that of *backward-* vs. *forward-looking* function of dialogue acts in DAMSL (Core and Allen, 1993). The *active* function of hints looks forward and seeks to help the student in accessing a further bit of information, by means of *eliciting*, that will bring him closer to the solution. The student has to think of and produce the answer that is hinted at.

The *passive* function of hints refers to the small piece of information that is provided each time in order to bring the student closer to some answer. The tutor *gives away* some information, which he has normally unsuccessfully tried to elicit previously. Due to that relation between the active and passive function of hints, the passive function of one hint class in the second dimension consists of hint categories that are included in the active function in its subordinate class.

2.3 Second Dimension

Domain-relation hints address the relations between mathematical concepts in the domain. An example of a relation that we have defined is *antithesis*, which captures that two concepts are opposites. For example, the element relation $\in$ is in *antithesis* to its opposite $\notin$. The passive function of domain-relation hints is the active function of domain-object hints, that is, they are used to elicit domain objects.

	active	passive
domain-relation	elicit-antithesis elicit-duality elicit-junction elicit-hypotaxis elicit-specialisation elicit-generalisation	give-away-antithesis give-away-duality give-away-junction give-away-hypotaxis give-away-specialisation give-away-generalisation
domain-object	give-away-antithesis give-away-duality give-away-junction give-away-hypotaxis give-away-specialisation give-away-generalisation	give-away-relevant-concept give-away-hypotactical-concept give-away-primitive-concept
inference rule	give-away-relevant-concept give-away-hypotactical-concept give-away-primitive-concept elaborate-domain-object	give-away-inference-rule
substitution	give-away-inference-rule elicit-substitution	spell-out-substitution
meta-reasoning	spell-out-substitution	explain-meta-reasoning
performable-step	explain-meta-reasoning confer-to-lesson	give-away-performable-step
pragmatic	ordered-list unordered-list elicit-discrepancy	take-for-granted point-to-lesson

Table 1: The taxonomy of hints.

Domain-object hints address an object in the domain. The hint *give-away-relevant-concept* names the most prominent concept in the proposition or formula under consideration. This might be, for instance, the concept whose definition the student needs to use in order to proceed with the proof, or the concept that will in general lead the student to understand which inference rule he has to apply. Other examples in the class are *give-away-hypotactical-concept* and *give-away-primitive-concept*. The terms hypotactical and primitive concept refer to the relation, based on the domain hierarchy, between the addressed concept and the original relevant concept, which the tutor is trying to elicit. The passive function of domain-object hints is used to elicit the applicable inference rule, and, therefore, is part of the active function of the respective class.

Inference-rule hints refer to the inference rules in the domain that need to be applied in the current proof. An additional active inference rule hint is *elaborate-domain-object*. It asks the student to elaborate on the domain object at hand in order to elicit the inference rule. The passive function, *give-away-inference-rule*, names the inference rule to be used. It is used to elicit the substitution of the rule for the problem at hand.

The active function of *substitution* hints consists of the passive function of the inference-rule hint plus the hint *elicit-substitution*. The latter asks the student to bind variables of the inference rule. The passive function *spell-out-substitution* explains to the student the way the substitution is done.

The active function of *meta-reasoning* hints is *spell-out-substitution*. It is, thus, the last hint that leads to the completion of the hints about the meta-reasoning employed. Therefore, its passive function explains the meta-reasoning employed so far.

The active function of *performable-step* hints consists of the passive function of meta-reasoning hints plus the hint *confer-to-lesson*, which points the student to the section in the study material where the answer can be found. Since there is no meta-reasoning left to be explained, the passive function is *give-away-performable* step.

Finally, the class of *pragmatic* hints is somewhat different from other classes in that it makes use of minimal domain knowledge. It rather refers to pragmatic attributes of the expected answer. The active function hints are *ordered-list*, which specifically refers to the order in which the parts of the expected answer appear, *unordered-list*, which

only refers to the number of the parts, and *elicit-discrepancy*, which points out that there is a discrepancy between the student's answer and the expected answer. The latter can be used in place of all other active hint categories. *Take-for-granted* asks the student to just accept something as a fact either when the student cannot understand the explanation or when the explanation would require making use of formal logic. *Point-to-lesson* points the student to the lesson in general and asks him to read it again when it appears that he cannot be helped by tutoring because he does not remember the study material. There is no one-to-one correspondence between the active and passive pragmatic hints. Some pragmatic hints can be used in combination with hints from other classes.

3 A Hinting Algorithm

A tutorial system ideally aims at having the student find the solution to a problem by himself. Only if the student gets stuck should the system intervene. There is pedagogical evidence (Chi et al., 1994; Rosé et al., 2001) that students learn better if the tutor does not give away the answer but instead gives hints that prompt the student for self-explanations. Accordingly, based on the work by Tsovaltzi (2001) we have derived an algorithm that implements an eliciting strategy that is user-adaptive by choosing hints tailored to the students. Only if hints appear not to help does the algorithm switch to an explaining strategy, where it gives away the answer and explains it. We shall follow Person and colleagues (2000) and Rosé and colleagues (2001) in calling the eliciting strategy *socratic* and the explaining strategy *didactic*.

To determine which parts of the solution need to be addressed the algorithm has to examine the student's answer. Therefore, we shall briefly present the student answer categories we use in the algorithm in Section 3.1 before we introduce the hinting algorithm in more detail in Section 3.2.

3.1 Student Answer Categories

The algorithm that determines if a hint is to be produced and chooses the hint takes into account the last answer the student has given. In particular, we need to categorise the student's answer in terms of its completeness and accuracy with respect to the expected answer. We say that an answer is *complete* if and only if all desired parts of the answer are mentioned. We say that a part of an answer is *accurate* if and only if the propositional content of the part is the true and desired one. Based on these notions, we define the following student answer categories:

Correct: The answer is complete and all parts are accurate.

Incomplete-Accurate: The answer is incomplete, but all parts that are there are accurate.

Inaccurate: The answer is complete or incomplete, but some parts of the answer are inaccurate.[1]

Wrong: The answer is incomplete and all parts are inaccurate.

We consider over-answering as several distinct answers, that is, if the student's answer has more parts than needed, these parts are considered as another answer, which can be categorised in turn.

3.2 The Algorithm

We shall now present an algorithm that implements the socratic strategy. In intuitive terms, the algorithm aims at having the student find the proof by himself. If the student does not know how to proceed or makes a mistake, the algorithm prefers hinting at the right solution in order to elicit the problem solving instead of giving away the answer. An implicit student model makes the algorithm sensitive to students of a different level by providing increasingly informative hints. Only if the hinting does not effect correct student answers after several hints does the algorithm switch to a didactic strategy, and, without hinting, explains the steps that the student cannot find himself. Nevertheless, the algorithm continues to ask the student for the subsequent step. If the student gives correct answers again and, thus, the tutor need not explain anymore, the algorithm switches back to the socratic strategy. In effect, hints are provided again to elicit the step under consideration.

[1] Note that this category is in fact an aggregation of several categories, which we need not distinguish for the purposes of this paper.

The Main Algorithm

The algorithm takes into account not only current and previous student answers and the number of wrong answers, but also the previous hints it produced both with respect to the hint category and the number of hints produced. The essentials of the algorithm are as follows:

1. if the proof is not completed
   then prompt for the next step
   else stop
2. analyse the student answer
3. if the student's answer is correct
   then accept the answer and go to step 1.
   else call the function `socratic` on the inaccurate and missing parts of the student's answer

The Function `socratic`

The bulk of the work is done by the function `socratic`, which we only outline here. The function takes as an argument the category C of the student's current answer. If the origin of the student's mistake is not clear, a clarification dialogue is initiated, which we shall not describe here. Note, however, that the function stops if the student gives the correct answer during that clarification dialogue, as that means that the student corrected himself. Otherwise, the function produces a hint in a user-adaptive manner.

The function `socratic` calls several other functions, which we shall explain subsequently.

Let H denote the number of hints produced so far and C_{-1} the category of the student's previous answer. The hint is then produced as follows:

Case $H = 0$
  if C is wrong or inaccurate then call `elicit`
  if C is incomplete-accurate
  then produce an active pragmatic hint
    {*that is, ordered-list, or unordered-list*}
Case $H = 1$
  if C is wrong
  then if C_{-1} is wrong or incomplete-accurate
    then call `up-to-inference-rule`
    if C_{-1} is inaccurate
    then call `elicit-give-away`
    if C_{-1} is correct
    then call `elicit`
  else call `elicit`
Case $H = 2$
  if C is wrong
  then if this is the third wrong answer
    then produce *explain-meta-reasoning*
    else if previous hint was an active substitution hint
      then produce *spell-out-substitution*
      else if previous hint was *spell-out-substitution*
        then produce *give-away-performable-step*
        else call `up-to-inference-rule`
  else call `elicit-give-away`
Case $H = 3$
  if C is wrong and it is at least the third wrong answer
  then produce *point-to-lesson* and stop
    {*The student is asked to read the lesson again. Afterwards, the algorithm starts anew.*}
  else produce *explain-meta-reasoning*
Case $H \geq 4$
  give away the answer and switch to didactic strategy; switch back after three consecutive correct answers with all counters reset
  {*After four hints, the algorithm starts to guide the student more than before to avoid frustration. If the student is able to follow again the tutor's plan for addressing the task, the algorithm switches back to the socratic strategy again and lets the student take over. If the student carries on giving correct answers the main algorithm guarantees that the tutor just accepts the answer and does not intervene further. Only if the student makes a mistake again will the hinting start anew with all counters reset.*}

After having produced a hint the function `socratic` analyses the student's answer to that hint. If the student's answer is still not right the function `socratic` is recursively called. However, if the student answers correctly and at least two hints have been produced the algorithm re-visits the produced hints in the reverse order to recapitulate the proof step and to make sure the student understands the reasoning so far. This is done by producing a sequence of active meta-reasoning hints, one for each hint that have addressed the current step of the proof, in the reverse order. If the active meta-reasoning hints get the student to say anything but the right answer, the algorithm produces an *explain-meta-reasoning* hint. This is done to avoid frustrating the student as his performance is poor.

The function `socratic` calls several functions, which we shall present now. The functions are self-explanatory.

Function `elicit`

if the student knows the relevant concept
then if the student knows the inference rule
  then call `up-to-substitution`
  else call `up-to-inference-rule`
else call `elicit-relevant-concept`

Function `elicit-give-away`

if previous hint was an active domain-object hint
then call `up-to-inference-rule`
else if previous hint was a passive

domain-object hint
then produce *elaborate-domain-object*
else if previous hint was
elaborate-domain-object
then call `up-to-substitution`
else produce *give-away-performable-step*
and *spell-out-substitution*
{*this is to explain the substitution*}

Function `up-to-inference-rule`
if previous hint was
give-away-hypotactical-concept
then produce *give-away-inference-rule*
else if previous hint was
give-away-relevant-concept
then produce *elaborate-domain-object*
else if previous hint was
elaborate-domain-object
then produce
give-away-hypotactical-concept
else produce
give-away-relevant-concept

Function `up-to-substitution`
if previous hint was *elicit-substitution*
then produce *spell-out-substitution*
else if right inference rule known
then produce *elicit-substitution*
else produce *give-away-inference-rule*

Function `elicit-relevant-concept`
if the student knows a concept that is related
to the relevant concept
then produce an active domain-object hint
else produce *give-away-relevant-concept*

4 An Example Dialogue

To elucidate how the algorithm for the socratic tutoring strategy proceeds, let us consider the following dialogue as it might occur between a tutoring system and a student. The tutoring system, denoted as tutor in the following, is supposed to teach the proof of the proposition $A \cap B \in \mathcal{P}(A \cup B)$, where $\mathcal{P}(X)$ stands for the powerset of the set X, that is, the set of all subsets of X, including the empty set and X itself. The proof is as follows: Obviously, $A \cap B \subseteq A \cup B$. Thus, the proposition follows by the definition of powerset.

When the algorithm starts, the counter H for the hints produced so far is initialised to 0. Since the proof is not completed, the tutor prompts for the first step by introducing the task.

Tutor (1): Prove that $A \cap B \in \mathcal{P}(A \cup B)$.

The expected answer is that by the application of the definition of powerset it suffices to show that $A \cap B \subseteq A \cup B$. However, the student answers:

Student (2): I don't know how to prove that.

This utterance is categorised as a wrong answer, since it does not contain any accurate part. Thus the algorithm calls the function `socratic`. Since $H = 0$, it calls the functions `elicit` and, thus, `elicit-relevant-concept` and eventually produces a hint of type *give-away-relevant-concept* (where powerset is the relevant concept) to elicit the inference rule (namely, the application of a definition).

Tutor (3): Think of using powerset.

With the production of this hint, the counter H is incremented by 1. However, this hint is not sufficient to help the student:

Student (4): How can I use it?

This again is categorised as a wrong answer. Thus, the function `socratic` is called recursively. Since $H = 1$, the function `up-to-inference-rule` is called, which produces a hint of type *elaborate-domain-object* to elicit the inference rule.

Tutor (5): What are the properties of powerset?

Again, the counter H is incremented by 1. The student answers:

Student (6): I don't know.

This answer is also considered as wrong. Since this is the third wrong answer and $H = 2$, the algorithm now produces an *explain-meta-reasoning* hint and explains the reasoning behind this step.

Tutor (7): You can start by applying the definition of powerset, which connects powerset to subset. This will simplify the problem.

The counter H is again incremented by 1.

Student (8): I don't understand.

This is again a wrong answer. Since $H = 4$, the algorithm produces a *give-away-performable-step* hint:

Tutor (9): According to the definition, the powerset of a set X is the set of all subsets Y of X. That is, if $Y \subseteq X$ then $Y \in \mathcal{P}(X)$. Now, you need to substitute X and Y with the right formulae from the proposition you want to prove.

Subsequently, the system switches to the didactic strategy, which we will not pursue in this paper. After having completed the explanation of this proof step, the system re-visits all hints produced by the socratic strategy while explaining the step, and makes the student aware of why they were produced. However, we will not go into such detail here, since it is not in the focus of this paper.

Instead, let us examine what happens if the student gives partial answers. Let us assume that the first hint uttered in (3) leads the student into the right direction:

Student (4'): I can try to apply the definition of powerset.

Since the student names the correct inference rule, but does not give the instantiation, this is considered as an incomplete-accurate answer. Hence, the algorithm calls the function `elicit`. Since the student already knows both the right relevant concept and the right inference rule, the tutor produces an *elicit-substitution* hint:

Tutor (5'): Can you spell out the application?

Student (6'): From the definition of the powerset follows that if $A \cap B \subseteq A \cup B$ then $A \cap B \in \mathcal{P}(A \cup B)$. That is, all I have to do is prove that $A \cap B \subseteq A \cup B$.

This is the correct answer, which also completes the first proof step. Since only one hint was produced, the recapitulation is omitted. Thus the algorithm proceeds with prompting for the next step.

5 Related Work and Discussion

Several other tutorial systems tackle hinting strategies as well. The BE&E project (Core et al., 2000) investigated multi-turn tutorial strategies in basic electricity and electronics. Some of these strategies are motivated by similar theoretical interests to the ones presented here. However, the number of strategies is small and no emphasis is given to the way information is made salient, which is the aim of our taxonomy. Moreover, there are no criteria, equivalent to our algorithm, for choosing one strategy over another.

Ms. Lindquist (Heffernan and Koedinger, 2000), a tutorial system for high-school algebra, also has some domain specific types of questions that resemble the BE&E strategies in form. Although there is some mention of hints, and the notion of gradually revealing information by rephrasing the question is prominent, there is no taxonomy of hints or any suggestions for dynamically producing them.

An analysis of hints can also be found in the CIRCSIM-Tutor (Hume et al., 1996a; Hume et al., 1996b), an intelligent tutoring system for blood circulation. Our work has been largely inspired by the CIRCSIM project both for the general planning of the hinting process and for the taxonomy of hints. CIRCSIM-Tutor uses domain specific hint tactics that are applied locally, but does not include a global hinting strategy that models the cognitive reasoning behind the choice of hints. We, instead, make use of the hinting history in a more structured manner. Our algorithm takes into account the kind of hints produced previously as well as the necessary pedagogical knowledge, and follows a smooth transition from less to more informative hints. Furthermore, we have defined a structured hint taxonomy with refined definition of classes and categories based on the passive vs. active distinction, which is similar to active-passive continuum in CIRCSIM. We have distinguished these from functions, which resemble CIRCSIM tactics, but are again more detailed and more clearly defined. All this facilitates the automation of the hint production.

AutoTutor (Person et al., 2000) uses curriculum scripts on which the tutoring of computer literacy is based. There is mention of hints that are used by every script. Although it is not clear exactly what those hints are, they seem to be static. More emphasis seems to be put on the pedagogically oriented choice of dialogue moves, prosodic features and facial expression features, but not on hints. In contrast, we have presented in this paper a hint taxonomy (cf. Section 2) and a tutoring algorithm (cf. Section 3) that models the dynamic generation of hints according to the needs of the student.

AutoTutor also uses a cycle of prompting-hinting-elaborating. This structure relies on the different role of the dialogue moves involved to capture the fact that the tutor provides more and more information if the student cannot follow the tutoring well. However, it does not provide hints that themselves reveal more information as the tu-

toring process progresses, which we have modelled in our algorithm for the Socratic teaching method (cf. Section 3). Thus, the student is not merely made to articulate the expected answers, as is the case in AutoTutor, but he is also encouraged to actively produce the content of the answer itself. Furthermore, the separation of the study material and the tutoring session facilitates the production of the answers by the student, since the tutor does not have to present the material and re-elicit it in one and the same session. The student is guided through making use of the study material that he has already read in order to solve the problem.

6 Conclusion and Future Work

There is psychological evidence that hinting in tutorial dialogue has a positive effect on the student's learning (Chi et al., 1994; Rosé et al., 2001). To take advantage of this effect, we have been developing a taxonomy of hints and a socratic algorithm for hint production in tutorial dialogues in mathematics. This algorithm should be easily adaptable to other tutoring domains as well.

Before we implement the algorithm in an intelligent system for tutorial dialogues, we chose to perform Wizard-of-Oz experiments to test the adequacy of the taxonomy and the effectiveness of the socratic algorithm. The experiments have already been completed and are currently being evaluated.

The socratic algorithm presented in this paper does not deal with the realisation of the hints. We are currently investigating the surface structure of hints and their generation. Moreover, the produced hints do not necessarily complete the tutor's dialogue turn. Further examination is needed to suggest a model of dialogue moves and dialogue specifications in our domain.

References

Michelene T. H. Chi, Nicholas de Leeuw, Mei-Hung Chiu, and Christian Lavancher. 1994. Eliciting self-explanation improves understanding. *Cognitive Science*, 18:439–477.

Mark G. Core and James F. Allen. 1993. Coding dialogues with DAMSL annotation scheme. In *AAAI Fall Symposium on Communicative Action in Humans and Machines*, pages 28–35, Boston, MA.

Mark G. Core, Johanna D. Moore, and Claus Zinn. 2000. Supporting constructive learning with a feedback planner. In *Proceedings of the AAAI Fall Symposium: Building Dialogue Systems for Tutorial Applications*, Falmouth, MA. AAAI Press.

Armin Fiedler and Helmut Horacek. 2001. Towards understanding the role of hints in tutorial dialogues. In *BI-DIALOG: 5th Workshop on Formal Semantics and Pragmatics in Dialogue*, pages 40–44, Bielefeld, Germany.

Neil T. Heffernan and Kenneth R. Koedinger. 2000. Building a 3rd generation ITS for symbolization: Adding a tutorial model with multiple tutorial strategies. In *Proceedings of the ITS 2000 Workshop on Algebra Learning*, Montréal, Canada.

Gregory Hume, Joel Michael, Allen Rovick, and Martha Evens. 1996a. Student responses and follow up tutorial tactics in an ITS. In *Proceedings of the 9th Florida Artificial Intelligence Research Symposium*, pages 168–172, Key West, FL.

Gregory D. Hume, Michael A. Joel, Rovick A. Allen, and Martha W. Evens. 1996b. Hinting as a tactic in one-on-one tutoring. *Journal of the Learning Sciences*, 5(1):23–47.

Natalie K. Person, Arthur C. Graesser, Derek Harter, Eric Mathews, and the Tutoring Research Group. 2000. Dialog move generation and conversation management in AutoTutor. In *Proceedings of the AAAI Fall Symposium: Building Dialogue Systems for Tutorial Applications*, pages 45–51, Falmouth, MA. AAAI Press.

Manfred Pinkal, Jörg Siekmann, and Christoph Benzmüller. 2001. Projektantrag Teilprojekt MI3 — DIALOG: Tutorieller Dialog mit einem mathematischen Assistenzsystem. In *Fortsetzungsantrag SFB 378 — Ressourcenadaptive kognitve Prozesse*, Universität des Saarlandes, Saarbrücken, Germany.

Carolyn P. Rosé, Johanna D. Moore, Kurt VanLehn, and David Allbritton. 2001. A comparative evaluation of socratic versus didactic tutoring. In Johanna Moore and Keith Stenning, editors, *Proceedings 23rd Annual Conference of the Cognitive Science Society*, University of Edinburgh, Scotland, UK.

Dimitra Tsovaltzi and Colin Matheson. 2002. Formalising hinting in tutorial dialogues. In *EDILOG: 6th workshop on the semantics and pragmatics of dialogue*, pages 185–192, Edinburgh, Scotland, UK.

Dimitra Tsovaltzi. 2001. Formalising hinting in tutorial dialogues. Master's thesis, Division of Informatics, University of Edinburgh, Scotland, UK.