Using ConceptGrid As an Easy Authoring Technique to Check Natural Language Responses

Dr. Stephen B. Blessing*
University of Tampa
401 W. Kennedy Blvd.
Tampa, FL 33606, USA
E-mail: sblessing@ut.edu
*Corresponding author

Mr. Shrenik Devasani
640 Epic Way 361
San Jose CA 95134
E-mail: dshrenik@gmail.com

Dr. Stephen B. Gilbert
Industrial & Manufacturing Systems Engineering
Iowa State University
Ames, IA 50011, USA
E-mail: gilbert@iastate.edu

Dr. Jivko Sinapov
Computer Science
Iowa State University
Ames, IA 50011, USA
E-mail: jsinapov@iastate.edu

Abstract: ConceptGrid provides a template-style approach to check natural language responses by students using a model-tracing style intelligent tutoring system. The tutor-author creates, using a web-based authoring system, a lattice-style structure that contains the set of required concepts that need to be in a student response. The author can also create just-in-time feedback based on the concepts present or absent in the student's response. ConceptGrid is integrated within the xPST authoring tool and was tested in two experiments, both of which show the efficacy of the technique to check student answers. The first study tested the tutor's effectiveness overall in the domain of statistics. The second study investigated ConceptGrid's use by non-programmers and non-cognitive scientists. ConceptGrid extends existing capabilities for authoring of intelligent tutors by using this template-based approach for checking sentence-length natural language input.

Keywords: intelligent tutoring; natural language; authoring tool; model-tracing; web-based authoring; just-in-time feedback; xPST.

Biographical notes: Stephen Blessing is an Associate Professor of Psychology at the University of Tampa. His main research interest is in authoring tools for intelligent tutoring systems, and he has co-edited a book on the topic. He has an interest not only in how ITSs are used in traditional classroom environments, but also how they may be used in informal learning environments as well. He is currently testing an iPad-based tutor in a children's museum.

Shrenik Devasani recently graduated with his MS in Human Computer Interaction and Computer Science from Iowa State University. His master's thesis described contributions to intelligent tutoring system authoring tools for non-programmers. Shrenik is passionate about educational technology and personalized online learning environments. He is now a Software Development Engineer at Amazon.com and is working on Amazon Silk, the browser on Kindle Fire.

Stephen Gilbert is Assistant Professor in Industrial & Manufacturing Systems Engineering and Associate Director of Iowa State's Virtual Reality Applications Center and its Graduate Program in Human Computer Interaction. His research focuses on intelligent tutoring systems. While he has built tutors for engineering education and more traditional classroom environments, his particular interest is their use in whole-body real-world tasks such as training for soldiers and first responders or for machine maintenance. His goal is to integrate ITSs into game engine simulations with multiple players and use biometric data as a partial cue to the learner's state.

Jivko Sinapov received the Ph.D. degree in computer science and human-computer interaction from Iowa State University (ISU) in 2013. He is currently a Postdoctoral Researcher and Instructor affiliated with ISU's Human-Computer Interaction program. His research interests include developmental robotics, robotic perception, autonomous manipulation, machine learning, and natural language processing.

Acknowledgements: Portions of this work were presented in a paper titled "Lattice-Based Approach to Building Templates for Natural Language Understanding in Intelligent Tutoring Systems" at the 2011 Artificial Intelligence in Education Conference in Auckland, New Zealand and in a paper titled "Evaluating ConceptGrid: An Authoring System for Natural Language

Responses” at the 2012 FLAIRS Conference in Marco Island, Florida. This work was done with the partial support of the U.S. Air Force Office of Scientific Research.

Using ConceptGrid As an Easy Authoring Technique to Check Natural Language Responses

There is broad interest in allowing students to respond to certain questions using natural language in a computer-based intelligent tutoring system (e.g., Graesser et al., 2004; VanLehn et al., 2002). It intuitively seems that such an approach would be more convenient to the student in many situations while also allowing a deeper assessment of their knowledge. For example, allowing students to type in their own response to a question such as, "What is the difference between nominal and ordinal style data?" in a statistics and research methods tutor provides a richer experience and a more stringent test of their knowledge than asking the same question but having students pick from a small set of choices. Achieving this richer experience requires solving the difficult task of automating the evaluation of the students' responses. Computer tutors are good at checking answers that are arrived at by formula, algorithm, or from pre-set choices. Judging the correctness of natural language, however, is much more challenging and what ConceptGrid attempts to make more tractable.

ITS researchers also have a broader interest in opening up the authoring process to a wider class of people, including educators that are not trained in designing intelligent tutoring systems (ITSs), non-cognitive scientists, and non-programmers (Murray et al., 2003). By making the creation of ITSs easier, more people will create ITSs and the systems will become more prolific. Many ITS systems have demonstrated impressive learning results (e.g., Graesser et al., 2004; Mitrovic et al., 2001; Ritter, et al., 2007; VanLehn et al., 2005), so broadening the base of systems will benefit students and other learners. With ConceptGrid the desire is to extend the ease of authoring into responses requiring natural language.

Related Work

Past intelligent tutoring systems have given participants the capability of responding in natural language. Two notable tutors that accept natural language responses are Why2-Atlas (VanLehn et al., 2002) and AutoTutor (Graesser et al., 2004). Students in Why2-Atlas respond to qualitative physics problems using natural language. For example, the tutor might pose a problem about what happens if a person on a quickly descending elevator lets go of an object (see Jordan et al., 2006). Students need to type out their responses in plain English, and the tutor checks their answer based on a first-order predicate logic analysis of the sentences. Why2-Atlas uses a variety of techniques to deeply understand what the student typed, and then has a sophisticated method for managing the dialogue interaction with the student. This allows Why2-Atlas to check student answers at a very precise level of detail and follow-up with very targeted questions for the student to address.

AutoTutor, another tutor that accepts natural language, has been deployed in a variety of domains, such as physics, computer literacy, and critical thinking. Unlike Why2-Atlas, AutoTutor primarily uses Latent Semantic Analysis (LSA; Landauer et al., 1998) to check student responses (Graesser et al., 2000). While using LSA might make encoding ideal responses less time consuming, it limits the sophistication with which it can check actual student responses. For example, LSA largely ignores word order, and it cannot always recognize negation. Furthermore, LSA scores the actual student answer against the ideal set of answers, and so might not be able to indicate what exactly is wrong with the student response. However, as mentioned above, AutoTutor has been successful in assisting student learning in a variety of different domains.

Other tutors that support natural language dialog with students also exist, such as Ms. Lindquist for algebra symbolization (Heffernan and Koedinger, 2002), CIRCSIM Tutor for

medical word matching (Glass, 2001), and Summary Street for writing (Wade-Stein and Kintsch, 2004). These use a combination of techniques such as those discussed above. All of these tutors point to the potential power of enabling students to respond in natural language, possibly to support justifying numeric answers. Aleven and colleagues (1999) required students to provide reasons for steps taken in a geometry tutor. While their answers were not natural language in a strict sense, students had to select their reasons from a glossary of known choices, providing the reasons led to better learning. Students who did such explanations learned more than those who did not. Perhaps the most basic cause for why having students explain their reasoning, giving their answers in natural language, is the self-explanation effect (Chi et al., 1989). Students who explain their thinking, those who connect consciously one idea to the next, perform better than those who do not. Computer tutors that have students explain their reasoning are making use of this effect. Tutors should capitalize on this finding by accepting natural language answers.

Concerning the broader interest in opening up the authoring process of intelligent tutors to a wider class of people, a number of researchers are now working towards that goal with a number of different types of ITSs. A volume by Murray and colleagues (2003) contains description and discussion on some of these efforts; also see the *International Journal of Artificial Intelligence*, volume 19, number 2, an issue dedicated to authoring tools for intelligent tutors. Furthermore, Dolog and colleagues (2007) survey a number of research projects concerning the authoring of adaptive systems. As one specific example, Munoz and Ortigosa (2007) discuss a method to allow instructor-authors to easily create courses based on adaptive hypermedia.

However, none of these systems address the issue of how to easily author natural language responses. Why2-Atlas had an authoring system, but it is not clear how usable it was by

someone not familiar with programming concepts. With Autotutor's use of LSA, there is no true authoring within it. What might come closest to ConceptGrid is using regular expressions (i.e., regex) to perform pattern matching. Brown and Hardistry (2007) describe a tutor that instructs students on such expressions. With limited exposure (one class period), students who used the regex tutor performed better at simple problems involving regular expressions than control students. Additionally Blackwell (2001) discusses different techniques, including graphical ones, to assist in the creation of regular expressions. However, these techniques at pattern matching are clearly not meant to provide validation for the types of natural language expressions, clauses, and sentences that ConceptGrid is meant to validate. ConceptGrid serves to fill a particular and novel niche in that regards.

As stated previously, ConceptGrid was designed to be usable by non-programmers, sometimes referred to as end-user programmers (Scaffidi et al., 2005). However, like any computer-based authoring tool, it still requires computational thinking (Wing, 2006). To accommodate this, ConceptGrid's user interface was designed in the spirit of visual programming tools such as Scratch (Maloney et al., 2008), Alice (Pausch et al., 1995), and LabView (Wells and Travis, 1996), in which non-programmers can create algorithms by chaining together atomic components that can be triggered to execute and take actions on objects represented elsewhere in the interface. This approach was also taken by Aleven and colleagues (2009) in their CTAT tool for authoring model-tracing intelligent tutors using behavior graphs. Nardi's review of visual programming literature (Nardi, 1993) notes that while this graphical approach is not always more effective than other interaction designs, it has strong advantages for representing abstraction.

ConceptGrid is part of a larger project, the Extensible Problem Specific Tutor (xPST). xPST allows non-programmers and non-cognitive scientists to create model-tracing ITSs (Blessing et al., 2009). Authors create the instruction in an easy-to-use text-based authoring environment, and xPST contains the mechanisms that allow the instruction to be displayed in a variety of different student interfaces, such as existing websites, Paint.NET, and the Torque game engine. ConceptGrid produces output compatible with xPST, but could be extended to work in the context of other authoring systems whose designers desire to accept natural language input. More information concerning both ConceptGrid and xPST can be found at http://xpst.vrac.iastate.edu.

A Way to Ease the Authoring of Natural Language Responses

ConceptGrid tutors on short natural language responses, the typical length being a sentence or two. It is designed to evaluate semi-structured natural language, meaning text in which you can expect to find certain words, phrases, and known types of sentence structures. Experience building ITSs has demonstrated many instances in which it would be helpful to provide an opportunity to provide this kind of tutoring, such as for a definition or comparison between ideas. The ConceptGrid author creates templates that match expected phrases within the student's answers to achieve this.

Figure 1 illustrates this method for a question that might be found in a statistics and research methods course, "What two things must be true for the mean to be preferred over mode or median?" The instructor expects a correct student response to contain these two concepts: 1) the data must be normally distributed, and 2) the data must be either interval or ordinal. Within ConceptGrid, the author creates a set of templates to recognize each concept. A student can phrase these concepts in a variety of different ways, and either concept can be mentioned first.

For instance, both of these responses would be correct: 1) “The data must be normally distributed, and their type must be either interval or ratio,” and 2) “The numbers need to be either ratio or interval, and the distribution needs to be a bell curve.” These responses are both shown in Figure 1. In ConceptGrid, one set of templates would correspond to the different ways of indicating the data are normally distributed, and another set of templates would check to make sure the student mentions the correct measurement scales. ConceptGrid works best when the author narrows down the key phrase or idea that epitomizes the concept. Figure 1 demonstrates this. The templates that check for some statement about normality check narrowly for either “normal distribution” or “bell curve.” If the student uses any of those phrases in the answer, they should probably get credit for knowing that part of the answer. Given how ConceptGrid checks for word variants and misspellings, as will be described below, “normally distributed” can match to “normal distribution” depending on parameters. That is, ConceptGrid is more powerful than simple substring matches. The second set of templates then ensures that the words “interval” and “ratio” appear in the answer, in either order. (As an alternate approach, one template could just check for “interval,” with the author adding a third template to check for “ratio.”)

In addition to defining the templates for a correct solution, authors can also define templates that would indicate buggy knowledge (Brown and Burton, 1978) for which the author might want to provide remediation. For instance, the author could add a buggy third concept to Figure 1 that checks to see if the student typed something that would match “skewed distribution” and a buggy fourth concept that would match “nominal” or perhaps “ordinal” answers. ConceptGrid can then offer feedback to the student concerning these common misconceptions (described further below).

This template method for ConceptGrid balances the easy approach present in many systems (e.g., allow a test-maker to identify a list of words or phrases that would be acceptable for an answer) with the much more sophisticated approach of a system like Why2-Atlas where the author produces detailed code in order to indicate a proper response. The easy "list of phrases approach" can be learned in minutes by any interested party, but lacks power. The sophisticated approach can be much more flexible and powerful in how it checks student responses, but requires much training, even by those already knowledgeable about linguistic representations and cognitive science. ConceptGrid strikes a good balance, allowing some amount of expressiveness found in more sophisticated techniques while remaining easy to learn.

Before describing the details of ConceptGrid, Figure 2 shows the overall functional process behind the ConceptGrid approach. First, using the web-based tool, an author creates the tutor. At instruction time, the learner will input their response to a question. That response is checked against the templates as mentioned in the example above. The tutor chooses the feedback based on how these templates match, and finally the tutor presents the feedback to the student. Each of these functional steps is described in more detail in the next section.

ConceptGrid Details

Templates within ConceptGrid contain one or more atomic checktypes. A checktype is a function that tests for the presence or absence of a particular set of words. The current system contains the checktypes seen in Table 1. Each checktype in a template processes text independent of other checktypes. To provide additional information beyond what is contained in Table 1, the Levenshtein(*n, wordList*) checktype performs fuzzy string searching and tries to find a string that matches a pattern approximately, based on Levenshtein distance (Levenshtein, 1966). This distance measure equals the number of single-character primitive operations

(insertion, deletion and substitution) required to change one word to another. For example, the Levenshtein distance between “normal” and “normally” is 2, whereas the distance between “distributed” and “distribution” is 3. The argument *n* indicates the maximum of number of operations that can be performed. The NoVowels(*wordList*) checktype is an alternative to Levenshtein that removes vowels and then checks the word; that is, it returns true if a literal match is found, after ignoring vowels, with any of the words in word list. In addition to the word matching checktypes (Exact, Levenshtein, NoVowels, and Synonym), there are two checktypes that provide additional power to the author. The Not(*n*, *direction*) checktype checks for negation. It ensures that the *n* words appearing to the left or right (specified by the parameter *direction*) of a specific word do not contain the words present in the list below. The Any(*m*, *n*) checktype matches a sequence of words that is at least *m* words and at most *n* words long. For example, to recognize the phrase, “the data must be normally distributed,” an author might create a ConceptGrid template that looks for phrases in which the words “data” or “numbers” occur within 1-5 words of the word “normally,” which in turn should be next to “distributed” or “distribution” (see Figure 3).

The manner in which the checktypes work is a key feature of ConceptGrid. The implicit sequencing of checktypes in the lattice approach means that the resulting templates are finite parsers. That is, progress through the lattice corresponds to progress left-to-right in processing the input. The templates are represented internally as and-or trees, which are reductions into checktypes. The algorithm involves a combination of recursion and memoization to efficiently process the input. Since the algorithm might need to backtrack many times, memoization helps speed up the processing by having function calls avoid repeating the calculation of results for

previously processed inputs. ConceptGrid's approach is focused on words and numerical analysis, rather than grammar and logic and is hence non-structural.

A complete analysis of the expressive power of ConceptGrid as a system for recognizing sets of language formalisms has not been conducted. However, it is worth noting that while a ConceptGrid template does not have the full power of a regular expression, since templates cannot contain words of arbitrary length, such as strings that match the regular expression [A-Za-z], ConceptGrid templates would be classified as a more expressive, context-sensitive grammar rather than a context-free or regular grammar, since the Not checktype allows left-right position-based structures. Higher expressivity within a language usually leads to a more complex parsing process, and given that the goal of ConceptGrid is to be easily parsed by tutor authors, the current level of expressive power seems sufficient.

ConceptGrid allows for the creation of these concept templates through the web-based graphic user interface (GUI) shown in Figure 3. This interface is available for use at http://xpst.vrac.iastate.edu, and the source code is available to interested researchers. In this "lattice," tutor authors can type a sample student response and then edit a lattice automatically produced from that response, or provide the dimensions of the template to define the initial template size. Once defined, rows and columns can be added or deleted from the template by clicking the appropriate "+" to add a row or column or "X" to delete a row or a column. The checktype for a column (i.e., a word position in the template) is chosen from the drop-down menu. The arguments for the chosen concept are entered to the menu's right, and the relevant words are listed beneath the checktype, one per row, if there are multiple words that might appear in that position.

The template that appears in Figure 3 checks for one of the possible answers to the question featured in Figure 1, “What two things must be true for the mean to be preferred over mode or median?” The two concepts of normality and interval or ratio data are part of the answer, and the template shown in Figure 3 relates to normality. The template involves four atomic checktypes. The first checks for the strings “data,” “numbers,” or “distribution” (that could differ by a Levenshtein distance of 1, so “dta” and “number” would also qualify), the next checks for a run of between 1 and 5 words of any kind, and the last two checks make sure the phrase ends in something close to “normal distribution” or “bell curve.” All word options in the columns are possible, so technically “bell distribution” would also be accepted, though unlikely. Note that these templates can match any subpart of the total student input, and do not have to match just the whole of the student input. As long as the pattern contained within the template matches any part of the student input, then that concept is considered to have matched. A second template could be created for this concept to handle alternative phrasings, or perhaps to include a Not checktype to rule out phrases like “the numbers don’t form a bell curve.” These templates would have to be created according to the author’s experience, until real student responses can be examined to see how students naturally answer these sorts of questions.

Tutor authors can define as many concept templates as needed to adequately check a student response. Once all concepts and templates within those concepts have been defined, the tutor author fills out the Feedback Table, a ternary truth table specifying feedback to give the students based on their input, given the presence or absence of the relevant concepts in their answer. For example, consider the question that was posed in Figure 1, “What two things must be true for mean to be preferred over mode or median?” Recall the two concepts the instructor desired in a student answer, one being the normality of the data, and the other the type of

variable being investigated. Figure 4a shows a Feedback Table for this example based on these two concepts. Each row in the Feedback Table corresponds to a possible state of a student answer, given the presence or absence of concepts. The green checks in the table correspond to when the concept is present in the student input, and the red X's to when the concept is absent. For example, the author might understand the first two rows of the table as, "If the Normality concept is present, and the DataType concept is present, then the answer must be right, so give the feedback, "Good answer!" If the Normality concept is there, but the student forgot the DataType concept (red X), then give the feedback that affirms the answer for normality but asks for more information. In addition to the present and absent options, there is also a third option (indicated by a yellow hyphen icon) to indicate that it does not matter if the concept is present or absent for the system to consider if the student input matches that state (i.e., ignore, or "don't care"). The user interface allows the user to cycle between all three options by clicking on the corresponding icon.

Each student answer is run through the concepts, and then the matching row or rows are found in the Feedback Table to provide feedback to the student—perhaps a statement of correctness, or maybe a statement indicating that a concept is missing, or a concept is there but should not be. Figure 4b illustrates the example when the author implemented a buggy third concept discussed above that checks to see if the student thought the distribution had to be skewed. In the case of that fifth row, an author might understand the row to mean, "If the student response contains the skewed concept, then I don't care whether or not it talks about Normality or DataType (yellow hyphens). I need to address the skewed idea first with this feedback." If multiple feedback responses in the Feedback Table match the response, the feedback of each response is concatenated.

With regards to the lattice structure of ConceptGrid discussed previously and shown in Figure 3, its design was based on the English structure of reading left to right (the columns of the lattice) and the common use in English narrative of sequential bulleted lists for alternatives (the rows of the lattice). This structure, including its affordances for inserting and removing rows and columns, is somewhat familiar for users of spreadsheets, despite the different content of the cells. The ternary logic structure that underlies the Feedback Table (Figure 4) is more abstract, and represents a set of logical decisions graphically. Both of these structures graphically represent to the author the underlying logic within ConceptGrid. As has been demonstrated by previous research on effective representations of decisions, particularly in the software LabView, graphics have been shown to be particularly effective for non-programmers (Kiper et al., 1997).

The approach to evaluating ConceptGrid is described below. Experiment 1 ensured the efficiency of the approach. Using data collected in a real-world setting, the experiment evaluated how well ConceptGrid could score student answers when ConceptGrid was used by an experienced user, one versed in cognitive science and programming. The student answers were obtained in an actual tutor within the domain of statistics. In Experiment 2, people who were both non-programmers and non-cognitive scientists used ConceptGrid to evaluate answers provided by students to questions an actual instructor might ask on a quiz or homework exercise, again in the realm of statistics.

Experiment 1

The first experiment served as a test of feasibility of the lattice-style approach of ConceptGrid as an effective natural language analysis tool for student answers. A separate project by Maass and Blessing (2011) using xPST to develop a college-level statistics tutor was used to collect a corpus. This project, called xSTAT, gave students web-based homework

problems in which they evaluated and interpreted *t*-tests. The final step in these data-driven, real-world based problems was to interpret the results of the *t*-test, which entailed stating in one or two sentences whether or not the null hypothesis should be rejected or not, and then to conclude with a statement of how the independent variable affected the dependent variable. For instance, one question contained data about the effects of music on learning, and the student had to interpret the results of the *t*-test by writing something like, “Reject the null hypothesis. There is a significant difference in memory recall between the rock music and no music conditions” in a free-response text box. There are numerous variations the student could type as an answer to these questions, such as, “A significant difference exists between the no music and rock conditions upon memory recall. The researcher should reject the null hypothesis.” All of these variations should be accepted. This answer has five separate concepts that must be examined: 1) a statement of rejection of the null hypothesis; 2) the significance of the results; 3) ensuring that the statement concerning the null hypothesis matched that of the statement of significance 4) a mention of the independent variable; and 5) a mention of the dependent variable. Experiment 1 was designed to ensure that an experienced ConceptGrid user could create the templates that adequately evaluated the responses to this style of question.

Participant

The second author of this paper, ConceptGrid’s programmer and co-creator, used ConceptGrid to create templates for the student answers concerning the interpretation of *t*-tests. The participant was enrolled during this research in a Master’s program in Human-Computer Interaction and was an accomplished programmer with some cognitive science knowledge.

Materials and Procedures

The corpus that the author's ConceptGrid was checked against was created through the xSTAT project. For one homework assignment in a college-level statistics course, students used xSTAT to solve six real-world problems (i.e., in each problem a data set was provided to students, and they had to use SPSS to solve the numeric aspects of the problem) that required some type of *t*-test to solve. The last question in each problem asked for the interpretation of the test, as exemplified above. For purposes of the xSTAT project, half of the students used a version of xSTAT that provided feedback for each step of the problem (problems contained between seven and nine sub-answers; all except for the last answer could easily be checked by comparing a numeric answer or a selection from a radio button or pop-up menu). The other half of participants received no feedback. For the tutored version of xSTAT that checked the last question of these problems (i.e., the free response interpretation question), xPST code was directly entered without benefit of ConceptGrid's user interface that provides the benefits of a visual programming editor. This non-lattice approach had to be done by a cognitive scientist and programmer (the first author) who found it difficult to do. This challenge provided impetus for the ConceptGrid approach.

Forty-one students participated in the xSTAT project. As a group they received 233 total multi-step problems from the set of six problems (there is no record for 13 students of a sixth problem; either a database error occurred or the students closed the web browser before the sixth problem saved). The last step of each problem contained the natural language response. Because the students in the tutored condition might attempt this last step multiple times on the same problem, the corpus contained 554 unique responses. An instructor and a teaching assistant scored the responses based on the presence or absence of the five concepts listed above in the example (rejection, significance, the match between these two, and the independent and

dependent variables). The participant for this experiment then used ConceptGrid to produce templates, attempting to match the way the instructor and teaching assistant scored the answers.

Results

The participant for this experiment used ConceptGrid to produce a total of 22 templates to check student answers. Ten of these templates checked for the rejection, significance, and match between these two across all six problems. The remaining 12 templates checked specifically for the independent and dependent variables in those problems.

Table 2 displays the results for how these templates matched across the 554 responses. For three of the concepts of interest (rejection, significance, and match between these two), the correct answer depended on the results of the calculations. For half the problems, the student should reject the null hypothesis (and with corresponding adjustments for significance and the match), and for the other half of the problems the student should fail to reject the null hypothesis. The table displays how well the ConceptGrid template matches those two separate outcomes with both true positives and true negatives. Also displayed in the table is the overall accuracy of the templates, the sum of true positives and true negatives divided by the total number of responses, plus an accounting of the false positives, when the template matched but should not have, and false negatives, when the template did not match but should have. Given that this approach uses atomic components within a template that are somewhat similar to regular expression matching, it may be surprising to see the number of false negatives in matching some student responses. However, this approach is more expressive, as it checks responses by looking for smaller concepts and key phrases with the help of checktypes rather than literal word matching. For example, one student typed, “Fail to reject the null hypothes [sic]. There is an increase between males and females but not a significant one.” In part, this response resulted for

its particular question in a true positive for the rejection-fail concept despite the misspelling (and a true-negative for the rejection-true concept) and a false negative for the significant-fail concept (the human graders considered it correct, but it did not match the ConceptGrid template).

Discussion

The overall average accuracy of .97 was satisfying, and this result shows the overall efficacy of the ConceptGrid approach at representing the knowledge of the instructor accurately within a tutor for natural language responses to this particular problem. Unfortunately it is difficult to give an overall time estimate of how long it took the participant to create all the templates, as he was fixing bugs and making efficiencies in the implementation of ConceptGrid as he went along. However, the effort was ultimately not that great to create the 22 templates using the ConceptGrid interface.

The participant here was a computer programmer and had some knowledge of cognitive science. Even if the tool was only usable by that class of person, it would still have value within the authors' research group, and the approach might be attractive to other research groups creating similar tutors. However, the intent was to develop a tool usable by non-programmers and non-cognitive scientists, so that the authoring process for this class of tutor could be available to wider variety of people. Experiment 2 was designed to examine the usability of ConceptGrid by non-programmers.

Experiment 2

Experiment 2 had a similar structure as Experiment 1. Participants played the role of tutor authors using ConceptGrid to check natural language answers from an existing corpus. This corpus was obtained from students doing an actual homework assignment in a college-level statistics class. Unlike in Experiment 1, the participants in Experiment 2 were not programmers,

and neither were they cognitive scientists. This experiment, then, examined whether ConceptGrid was usable by those outside of a research lab.

Participants

Two current instructors of a college-level statistics course participated as tutor authors for this study. Both had taught statistics multiple times in the past. While both had Ph.D.'s within psychology, neither had cognitive psychology or computer science as their specialty, nor used a symbolic processing language like R or command line SPSS to perform statistics. Neither received compensation for their participation. In addition, the first author of this paper served as a third participant with intermediate skill level, as he understood the design intent of ConceptGrid, but had never used it to score actual student answers.

Materials and Procedure

The participants created tutoring using ConceptGrid around statistics-based content. Six questions were created that a student halfway through his or her first semester in an introductory statistics course should know. After initial construction of the question set, the set of six questions was finalized in consultation with the participants to confirm that these were questions that a majority of their students should know. The concepts for each question were derived from these consultations with the participants. The nature of the questions led to broad agreement as to what constituted a good answer. In addition participants created a ConceptGrid for a seventh question, one of the six used in Experiment 1, where the answer was to write a conclusion statement for a hypothesis test. The final list of seven questions is shown in Table 3. As can be seen, the questions are in roughly increasing order of complexity, from one where there is a single correct answer, to ones where there are multiple parts (i.e., concepts) but limited answers, and ones with multiple parts and open-ended phrasings. Each participant's task was to use the

ConceptGrid website to create seven ConceptGrids, one for each question. Each ConceptGrid could have as many templates as the participants desired to address each question's concepts.

The college instructors learned about ConceptGrid in one 45-minute face-to-face training session conducted by the first author of this paper. A four-page document was given to the instructors that contained an overview of ConceptGrid, login procedures for the ConceptGrid website, interface instructions on creating concepts and using the Feedback Table, and the list of seven questions to be tutored. The 4-page instruction set contained one simple ConceptGrid example (one that involved two concepts about Christopher Columbus), and during the short training session another example was developed. Neither example involved statistical content.

In order to test the ConceptGrids created by the tutor authors, a corpus of responses to these questions was needed. For Question 7, the 112 responses to that last question of the problem generated by real students for Experiment 1 were selected. To generate responses to the other six questions in this experiment, 87 current students from five different sections in a first semester college statistics course answered all questions using an online form. This generated another 522 responses. The responses from one of the sections were given to each participant as they were creating their initial ConceptGrids. This one section's worth of responses was meant to provide the participants at least a little insight as they started their ConceptGrids. Providing only a small subset of the data is a technique often used to train machine learning classifiers or neural networks so that the data are not overfit.

The participants had two weeks to complete their initial ConceptGrids after the training session. They were told to plan on it taking about two hours and were encouraged to email or ask any questions they might have as they went along. The authors spent about ten minutes with both novice participants answering questions of both a technical nature (e.g., what exactly the

argument for Levenshtein means) but also of a more conceptual nature (e.g., the best approach for making a template).

After completion of their initial set of ConceptGrids, the participants' solutions were tested against the entire corpus of 634 student responses. They were provided feedback concerning the accuracy of their ConceptGrids, and given the text of half the student responses their ConceptGrids miscategorized. Only half were used so that again they would not be tempted to overfit the data. They had two days to modify their ConceptGrids to see how much they might improve upon their accuracy rate, at which time their ConceptGrids were tested once more against the entire corpus. An overall high accuracy rate was expected, over .90, but decreasing with the increasing level of question complexity.

Results

Each participant created 14 ConceptGrids within the first iteration, and then updated them in a second iteration. The website kept track of how long participants worked on each of their ConceptGrids. The two beginning authors spent an average of 1.11 hours editing their first set of ConceptGrids. The intermediate author spent 0.39 hours. This is the time spent actually editing. The website also logged the total amount of time logged in, which would account for planning time. Unfortunately, one beginner user kept logged in while doing off-task behavior, so an accurate measure could not be obtained. For the other two users, there is a 2:1 ratio of total time to editing time. Precise timing data is not available for the second iteration due to a technical issue. Anecdotal evidence indicates users spent approximately 45 minutes on this second iteration.

A research assistant and the first author scored all the student responses for correctness. The seven total discrepancies between the raters (.01 of the corpus) were resolved by verbal

agreement. After each participant indicated he or she was done working on the initial ConceptGrid for each question, the ConceptsGrids were checked against the student responses. In such a way an accuracy score for each participant's ConceptGrids was obtained, indicating the percentage of time his or her ConceptGrids correctly rejected and correctly accepted the student responses. Table 3 displays the mean accuracy by concept. Note that for Question 7, the question from the previous experiment, the participants were asked only to check for the statement of rejection and significance and only one form was needed, as the correct answer was to fail to reject the null hypothesis.

Examining individual accuracy results, the two beginners scored an overall average of .78 and .88 on the first iteration, and then increased to .86 and .88 on the second iteration, respectively. The intermediate user went from .86 to .93. Questions 5 and 7 proved most difficult, due to the wide variability in student responses that was not reflected in the initial corpus subset that participants were given. Considering the improvement across all of the patterns that the participants authored, where the average went from .84 to .89, a significant difference with a non-trivial effect size was observed ($t(41) = 2.36$, $p = .023$, $d = 0.36$), demonstrating a marked learning trajectory.

The types of errors made by the authors' initial ConceptGrids are analysed below, and the total set of true positives, true negatives, false positives, and false negatives can be seen in Table 4. Like the expert user in Experiment 1, most errors were false negatives (545 in total, which equals .85 of all errors, or .13 of all checked responses). The false positives were fewer in number (99 total, .15 of all errors, .03 of all responses). The number of false negatives is higher than false positives, mainly because the ConceptGrid authors did not consider all possible representations of valid responses, particularly on certain questions. With practice ConceptGrid

authors will likely to design templates that cater to a wider range of responses. This process was observed with the revision data, as the number of false negatives decreased to 302 (.08 of all responses).

Participants tended towards authoring short templates. This indicates a strategy across questions to focus on a particular phrase that indicates student understanding and create a concept template for that phrase. This makes logical sense for some questions where the concept is a single word (e.g., all three concepts for the second question), but the participants adopted a minimalist approach for the other questions as well. The average number of atomic checktypes used per concept in the first iteration was 2.90, and that decreased slightly to 2.85 in the second iteration. All the different atomic checktypes were used at least once, but the Levenshtein and Any checktypes were used most often.

Discussion

These results for Experiment 2 were good, though the final accuracy was not quite above the desired .90 for the beginning authors. A slight decrease in accuracy with increasing complexity was observed, yet some of the more complex questions enjoyed a high accuracy. Also, some concepts enjoyed a noticeable improvement between attempts, indicating the usefulness of an iterative approach.

General Discussion

Both experiments show the efficacy of the ConceptGrid approach. As seen in Experiment 1, the creator of ConceptGrid enjoyed an accuracy rate above .97 in checking answers given in real-world problem scenarios. These were the type of non-trivial, multi-phrase answers that ConceptGrid was designed to tutor. Experiment 2 again provided evidence as to the worthiness of the approach, as not only did an intermediate-level user successfully use the tool, but non-

programmers and non-cognitive scientists also quickly learned how to use ConceptGrid in order to check answers given on a typical homework assignment. Taken together, this research indicates the utility of ConceptGrid in providing meaningful tutoring for natural language responses.

It will be worth evaluating how well ConceptGrid generalizes to additional situations and also to a wider range of users. ConceptGrid was used in the realm of statistics, to check not only basic statistical knowledge as it might appear in a homework or test situation (i.e., the items used in Experiment 2), but also to check the phrasing of a sentence as it might appear in an APA-style paper (the responses in Experiment 1). This result lends confidence that ConceptGrid would be useful across a wide variety of domains, such as economics (e.g., interpreting supply and demand curves), geometry (e.g., justifying answers via the appropriate theorem), or physics (e.g., providing qualitative answers to certain problem). If connected with a speech recognition system, ConceptGrid could also be useful in contexts in which spoken communication must follow strict syntax, such as in the military, for example, when a combat medic issues a MEDEVAC request. These requests follow a strict 9-line template, and a tutor using ConceptGrid could give feedback on the medic's consistency with the template. In a new research project, ConceptGrid will be used to create feedback for engineering undergraduates as they complete a reflective process of framing problems before they begin working on them. Answers to questions such as, “What are the critical forces in this problem?” and “What are the relationships between the unknowns?” should be addressable by ConceptGrid.

The fact that the novice users in Experiment 2 were able to use ConceptGrid effectively with minimal training and assistance demonstrates the viability of the approach with this class of instructor. This result demonstrates that non-programmers and non-cognitive scientists can create

this type of tutoring. Both novice participants in Experiment 2 reported that the system was easy to use, with a short learning curve. This is consistent with past work on the larger xPST system (e.g., Gilbert et al., 2011; Roselli et al., 2008). Novice users, who are non-programmers and non-cognitive scientists, can create meaningful and effective tutoring. Furthermore, the lattice-style approach that ConceptGrid is based on is not specific to xPST. Other authoring tools that aim at broadening the base of ITS authors could also use this approach to enable tutoring on similar natural language responses.

There are improvements that could be made within ConceptGrid, and these were particularly noted after experiences within Experiment 2. The feedback given for how well an author's ConceptGrids matched an existing corpus was not immediate. The author created the ConceptGrids and Feedback Tables, and then had to wait for the second author to manually run the corpus through what the author had done. This manual step should be automated. One could imagine loading a corpus into ConceptGrid, and then clicking a single button to check the corpus against the currently loaded ConceptGrids. This immediate feedback would be an obvious benefit to authors helping them to iterate faster.

Also of benefit to authors would be more understandable feedback regarding the correct and incorrect matches. The feedback currently consists of a grid of numbers, whose size was determined by the number of concepts and the size of the corpus. It is a lot of numbers (a large table of -1's, 0's, and 1's) and hard to decipher, particularly for non-programmers. The user design challenge is how to present all that data to be maximally effective to the author.

An automated approach could also be provided to developers and authors in constructing templates. After a corpus has been collected and scored by an instructor, it could then be used to generate the templates. That is, the patterns could be learned directly from the student responses.

This would still require a ConceptGrid author to structure and classify the templates into concepts, and produce a Feedback Table to provide appropriate responses to the student. However, this might serve to minimize the false positive and false negatives that are initially generated.

Lastly, even though the novice authors took to ConceptGrid without much instruction or oversight, there is room for improvement within the authoring tool itself. For example, confusion as to how the numerical argument to the Levenshtein checktype worked caused the acceptance of some false positives. If set too high, particularly for shorter words, that checktype will be too permissive in what it matches. A user interface simplification improvement might be to take away that numerical argument all together, and simply set the Levenshtein value dynamically as a function of the length of the corresponding word.

In conclusion, the two experiments demonstrate how effective ConceptGrid is as an authoring tool to check sentence-length natural language answers. The resulting tutoring has been deployed in actual classroom settings and has shown to improve student learning. This approach will prove useful across a variety of domains and authors.

References

Aleven, V., Koedinger, K., and Cross, K. (1999) 'Tutoring answer explanations fosters learning with understanding', in Lajoie, S.P. and Vivet, M. (Eds.), *Proceedings of the 9th International Conference on Artificial Intelligence in Education*, pp. 199-206.

Aleven, V., McLaren, B.M., Sewall, J., and Koedinger, K.R. (2009) 'A new paradigm for intelligent tutoring systems: Example-tracing tutors', *International Journal of Artificial Intelligence in Education*, Vol. 19, No. 2, pp. 105–154.

Blackwell, A. (2001) 'SWYN: A visual representation for regular expressions', in Liberman, H. (Ed.), *Your Wish is My Command: Programming by Example*, Morgan Kaufmann, San Francisco, pp. 245-270.

Blessing, S., Gilbert, S., Blankenship, L., and Sanghvi, B. (2009) 'From SDK to xPST: A new way to overlay a tutor on existing software', In Lane, H.C. and Guesgen, H. W. (Eds.), *Proceedings of the Twenty-Second International Florida Artificial Intelligence Research Society Conference*.

Brown, C. W. and Hardisty, E. A. (2007) 'RegeXeX: An interactive system providing regular expression exercises', *ACM SIGCSE Bulletin*, Vol., 39, No. 1, pp. 445-449.

Brown, J. S. and Burton, R. R. (1978). 'Diagnostic models for procedural bugs in basic mathematical skills', *Cognitive Science*, Vol. 2, No. 2, pp. 155-192.

Chi, M. T., 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.

Dolog, P., Kravcik, M., Cristea, A., Burgos, D., Bra, P., Ceri, S., Devedzic, V., Houben, G., Libbrecht, P., Matera, M., Melis, E., Nejdl, W., Specht, M., Stewart, C., Smits, D., Stash,

N., & Tattersall, C. (2007) ‘Specification, authoring and prototyping of personalised workplace learning solutions’, *International Journal of Learning Technology,* Vol. 3., No. 3, pp. 286-308.

Gilbert, S., Devasani, S., Kodavali, S., and Blessing, S. (2011) ‘Easy authoring of intelligent tutoring systems for synthetic environments’. In *Proceedings of the Twentieth Conference on Behavior Representation in Modeling and Simulation*.

Glass, M. (2001) ‘Processing language input in the CIRCSIM-tutor intelligent tutoring system’, in Moore, J.D. et al. (Eds.), *Proceedings of Artificial Intelligence in Education*, pp. 210-221.

Graesser, A., Wiemer-Hastings, P., Wiemer-Hastings, K., Harter, D., Person, N., & the Tutoring Research Group (2000) ‘Using latent semantic analysis to evaluate the contributions of students in AutoTutor’, *Interactive Learning Environments,* Vol 8., No. 2, pp. 129-147.

Graesser, A.C., Lu, S., Jackson, G.T., Mitchell, H., Ventura, M., Olney, A., and Louwerse, M.M. (2004) ‘AutoTutor: A tutor with dialogue in natural language’, *Behavioral Research Methods, Instruments, and Computers*, Vol. 36, No. 2, pp. 180-192.

Heffernan, N. and Koedinger, K. (2002) ‘An intelligent tutoring system incorporating a model of an experienced human tutor’, in Cerri, S.A. et al. (Eds.), *Sixth International Conference on Intelligent Tutoring Systems,* pp. 596-608.

Jordan, P., Makatchev, M., Pappuswamy, U., VanLehn, K., and Albacete, P. (2006) ‘A natural language tutorial dialogue system for physics’, in Sutcliffe, G. and Goebel, R. (Eds.), *Proceedings of the 19th International FLAIRS Conference*. Menlo Park, CA: AAAI Press, pp. 521-526.

Kiper, J. D., Auernheimer, B., and Ames, C. K. (1997) 'Visual depiction of decision statements: What is best for programmers and non-programmers?', *Empirical Software Engineering*, Vol. 2, No. 4, pp. 361-379.

Landauer, T.K., Foltz, P.W., and Laham, D. (1998) 'Introduction to latent semantic analysis', *Discourse Processes*, Vol. 25, Nos. 2-3, pp. 259-284.

Levenshtein, V. I. (1966) 'Binary codes capable of correcting deletions, insertions and reversals', *Soviet Physics Doklady*, Vol. 10, pp. 707-710.

Maass, J. K., and Blessing, S. B. (2011) 'Xstat: An intelligent homework helper for students', Poster presented at the *2011 Georgia Undergraduate Research in Psychology Conference*. 14 April 2011. Kennesaw, GA.

Maloney, J.H., Peppler, K., Kafai, Y., Resnick, M. and Rusk, N. (2008) 'Programming by choice: Urban youth learning programming with scratch', In *Proceedings of the 39th SIGCSE Technical Symposium on Computer Science Education*.

Mitrovic, A., Mayo, M., Suraweera, P., and Martin, B. (2001) 'Constraint-based tutors: a success story', *Engineering of Intelligent Systems*, pp. 931-940.

Munoz, F. and Ortigosa, A. (2007), 'Adaptive hypermedia in secondary schools: from the teacher to the student', *International Journal of Learning Technology*, Vol. 3, No. 3, pp. 309-323.

Murray, T. et al. (Eds.), (2003) *Authoring Tools for Advanced Technology Educational Software*. Kluwer Academic Publishers, Dordrecht, The Netherlands.

Nardi, B. A. (1993) *A small matter of programming: perspectives on end user computing.* MIT Press, Boston, MA.

Pausch, R., Burnette, T., Capeheart, A.C., Conway, M., Cosgrove, D., DeLine, R., Durbin, J., Gossweiler, R., Koga, S., and White, J. (1995) 'Alice: Rapid prototyping system for virtual reality', *IEEE Computer Graphics and Applications*, Vol. 15, No. 3, pp. 8-11.

Ritter, S., Anderson, J.R., Koedinger, K.R., and Corbett, A. (2007) 'Cognitive tutor: Applied research in mathematics education', *Psychonomic Bulletin and Review*, Vol. 14, No. 2, pp. 249-255.

Roselli, R.J., Gilbert, S., Howard, L., Blessing, S. B., Raut, A., and Pandian, P. (2008) 'Integration of an intelligent tutoring system with a web-based authoring system to develop online homework assignments with formative feedback', In *Proceedings of 115th Annual American Society for Engineering Education Conference*.

Scaffidi, C., Shaw, M., and Myers, B. (2005) 'An approach for categorizing end user programmers to guide software engineering research', *ACM SIGSOFT Software Engineering Notes*, Vol. 30, No. 4, pp. 1-5.

VanLehn, K., Lynch, C., Schulze, K., Shapiro, J.A., Shelby, R., Taylor, L., Treacy, D., Weinstein, A., and Wintersgill, M. (2005) 'The andes physics tutoring system: Lessons learned', *International Journal of Artificial Intelligence and Education*, Vol. 15, No. 3, pp. 1-47.

VanLehn, K., Jordan, P., Rosé, C. P., Bhembe, D., Böttner, M., Gaydos, A., Makatchev, M., Pappuswamy, U., Ringenberg, M., Roque, A., Siler, S., and Srivastava, R. (2002) 'The architecture of Why2-Atlas:A coach for qualitative physics essay writing', in Cerri, S.A. et al. (Eds.), *Sixth International Conference on Intelligent Tutoring Systems*, pp. 158-167.

Wade-Stein, D., and Kintsch, E. (2004) 'Summary Street: Interactive computer support for writing', *Cognition and Instruction*, Vol. 22, No. 3, pp. 333-362.

Wells, L. K. and Travis, J. (1996) *LabVIEW for everyone: Graphical programming made even easier*. Prentice Hall, Upper Saddle River, New Jersey.

Wing, J.M. (2006) 'Computational thinking', *Communications of the ACM*, Vol. 49, No. 3, pp. 33-35.

Table 1

ConceptGrid Checktypes

Checktype	Description
Any(*m*,*n*)	Allows a match of an arbitrary sequence of words between *m* and *n* words, when the words themselves are not important
Exact (*wordList*)	Returns true if a literal word match with any of the words in *wordList* is found
Levenshtein(*n*, *wordList*)	Returns true if the least Levenshtein distance between a word in *wordList* and matched word is <= *n*
Not(*n*, *wordList*, *direction*)	Checks to make sure a word in *wordList* does not appear in the sequence in the given *direction*; allows for a check of negation
NoVowels(*wordlist*)	Returns true if a literal match, after ignoring vowels, with any of the words in word_list is found
Synonym(*wordList*)	Uses the WordNet corpus to match synonyms, with an implied Levenshtein distance of 2

Table 2

Results of Experiment 1: Evaluation of ConceptGrid templates across 554 responses

Concept	True Positives	True Negatives	False Positives	False Negatives	Accuracy
Rejection-true	308	211	1	34	.94
Rejection-fail	246	297	6	5	.98
Significance-true	298	248	1	7	.99
Significance- fail	256	285	12	1	.98
Independent Variable	244	286	1	3	.99
Dependent Variable	217	331	4	3	.99
Conclusion-true	244	306	0	24	.96
Conclusion- fail	310	237	6	0	.99

Table 3

Questions and Accuracy Results of Experiment 2

Questions	First Iteration Accuracy	Second Iteration Accuracy
1. What statistic is the square root of variance?		
Concept 1.1 (“standard deviation”)	.98	1.00
2. What are the 3 main measures of central tendency?		
Concept 2.1 (“mean”)	1.00	1.00
Concept 2.2 (“median”)	1.00	1.00
Concept 2.3 (“mode”)	1.00	1.00
3. What is at least one aspect that differentiates a true experiment from a descriptive experiment?		
Concept 3.1 (“manipulation”)	.71	.74
Concept 3.2 (“control”)	.88	.85
Concept 3.3 (“causality”)	.98	.98
4. What two things must be true for *mean* to be preferred over *mode* or *median*?		
Concept 4.1 (“nomality”)	.78	.90
Concept 4.2 (“data type”)	.96	.96
5. What is the difference between nominal and ordinal style data?		
Concept 5.1 (“nominal”)	.62	.71
Concept 5.2 (“ordinal”)	.65	.73
6. What does parsimony mean?		
Concept 6.1 (“simplicity”)	.99	.98
7. What do you conclude based on these results? [this came after a hypothesis test had been conducted]		
Concept 7.1 (“rejection-fail”)	.57	.96
Concept 7.2 (“significance- fail”)	.63	.64
Overall	.84	.89

Table 4

Results of Experiment 2: Evaluation of ConceptGrid templates across 3 authors (Questions 1-6 had 87 responses; Question 7 had 112 responses

First Iteration Performance

Concepts	True Positives	True Negatives	False Positives	False Negatives	Accuracy
Concept 1.1	195	60	3	3	.98
Concept 2.1	222	39	0	0	1.00
Concept 2.2	207	54	0	0	1.00
Concept 2.3	210	51	0	0	1.00
Concept 3.1	39	147	18	57	.71
Concept 3.2	8	222	12	19	.88
Concept 3.3	84	173	0	4	.98
Concept 4.1	42	162	0	57	.78
Concept 4.2	111	140	0	10	.96
Concept 5.1	69	93	30	69	.62
Concept 5.2	80	90	9	82	.65
Concept 6.1	73	186	0	2	.99
Concept 7.1	74	117	0	145	.57
Concept 7.2	29	183	27	97	.63

Second Iteration Performance

Concepts	True Positives	True Negatives	False Positives	False Negatives	Accuracy
Concept 1.1	198	62	1	0	1.00
Concept 2.1	222	39	0	0	1.00
Concept 2.2	207	54	0	0	1.00
Concept 2.3	210	51	0	0	1.00
Concept 3.1	55	149	16	41	.74
Concept 3.2	15	208	26	12	.85
Concept 3.3	83	172	1	5	.98
Concept 4.1	79	155	7	20	.90
Concept 4.2	111	140	0	10	.96
Concept 5.1	79	105	18	59	.71
Concept 5.2	102	89	10	60	.73
Concept 6.1	75	182	4	0	.98
Concept 7.1	214	109	8	5	.96
Concept 7.2	36	179	31	90	.64

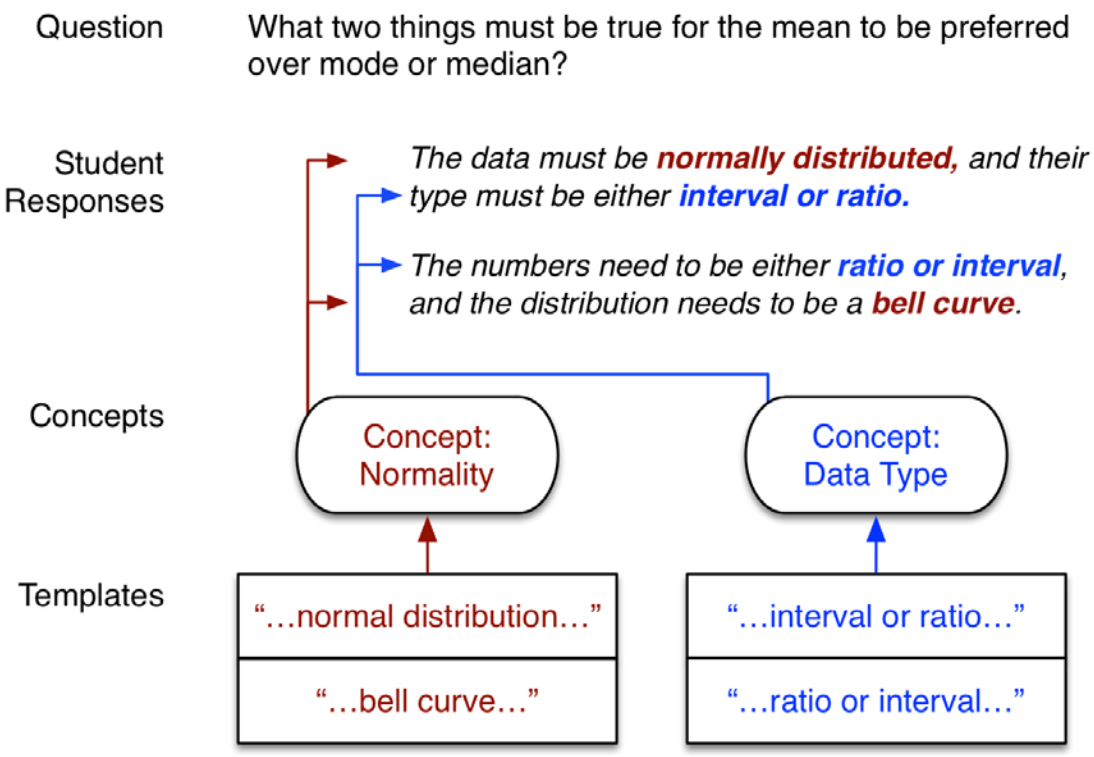

Figure 1. The natural language approach of ConceptGrid.

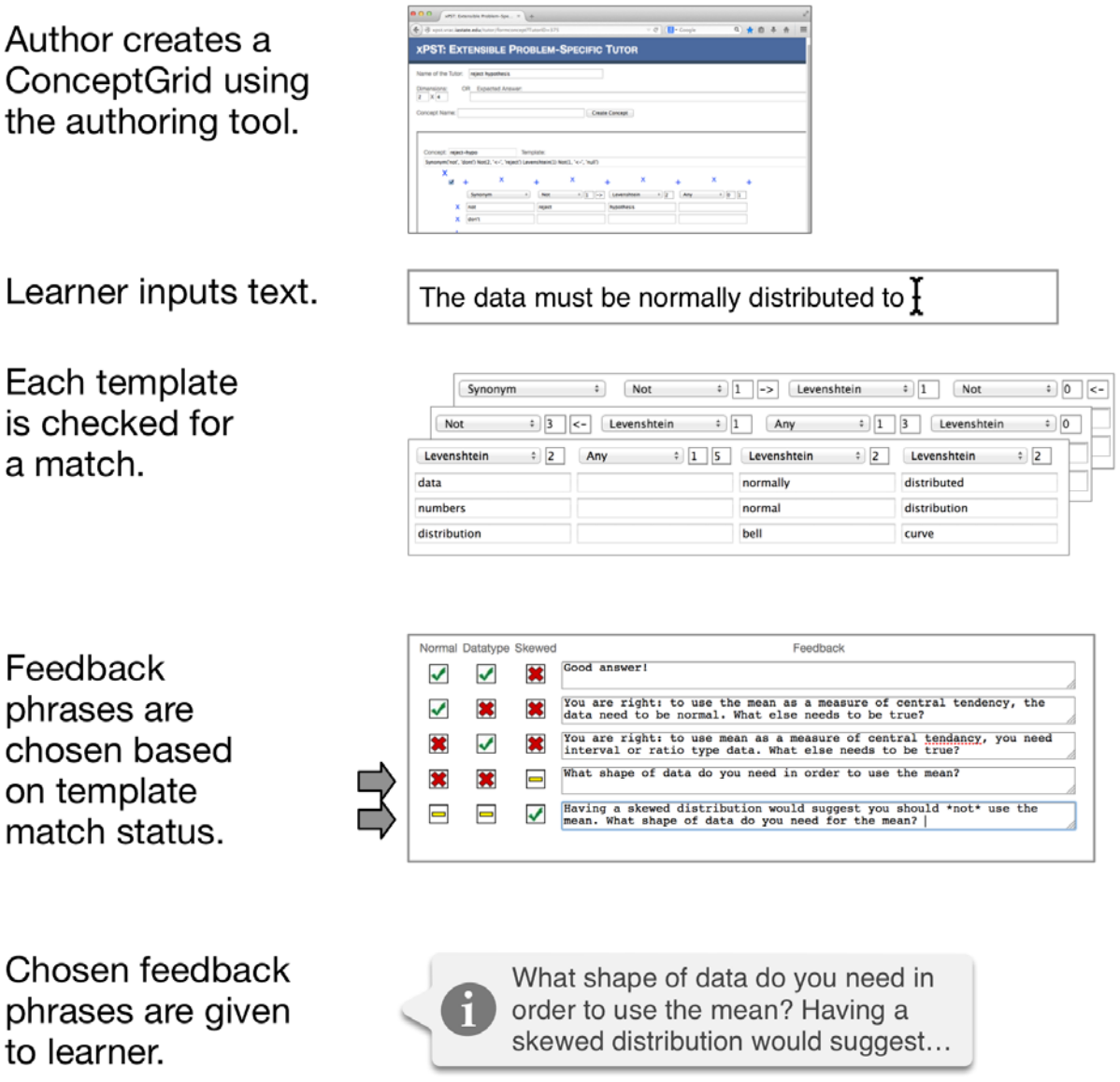

Figure 2. Process flow of ConceptGrid use.

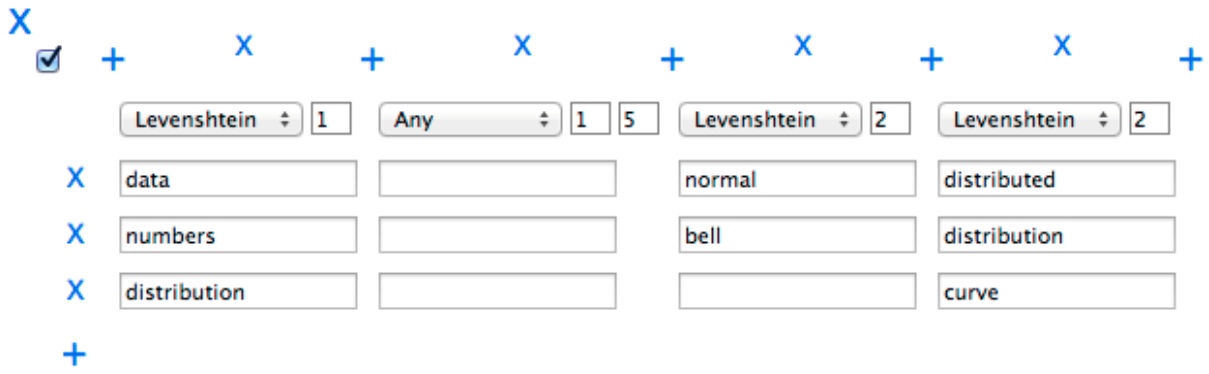

Figure 3. An example of a ConceptGrid template that would recognize phrases such as “data are normally distributed,” “numbers form a normal distribution,” or “distribution needs to be a bell curve.”

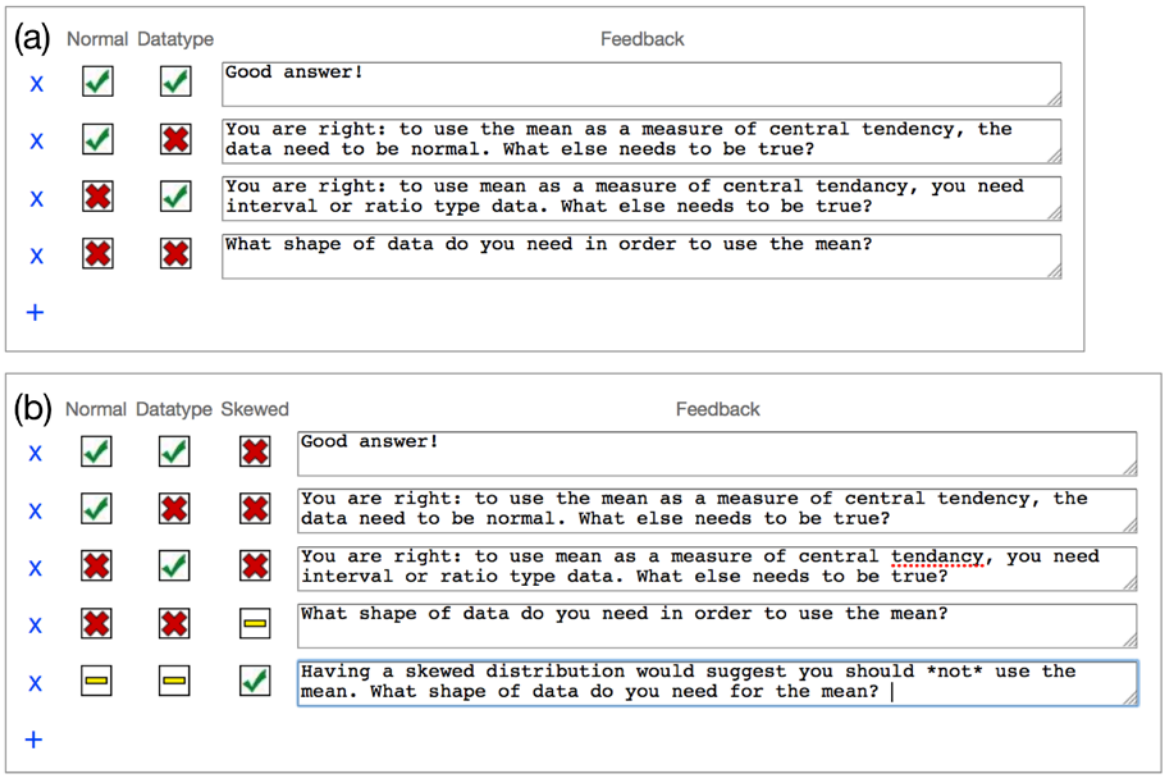

Figure 4. Examples of ternary Feedback Tables: (a) illustrates feedback possibilities for all true (green)/false (red) combinations of the two concepts Normal and Datatype; (b) illustrates the yellow hyphen to mean “ignore.”