Exploring the Relationship between Self-Efficacy and the Effectiveness of Tutorial Interactions

Joseph B. Wiggins[1], Joseph F. Grafsgaard[1], Christopher M. Mitchell[1], Kristy Elizabeth Boyer[1], Eric N. Wiebe[2], James C. Lester[1]

[1]Department of Computer Science [2]Department of STEM Education
North Carolina State University, Raleigh, NC, USA
{jbwiggi3, jfgrafsg, cmmitch2, keboyer, wiebe, lester}@ncsu.edu

Abstract. In recent years, significant advances have been made in intelligent tutoring systems, and these advances hold great promise for adaptively supporting computer science (CS) learning. In particular, tutorial dialogue systems that engage students in natural language dialogue can provide rich, adaptive interactions. One promising approach for increasing the effectiveness of these systems is to adapt not only to problem solving performance, but also to a student's incoming characteristics. *Self-efficacy* refers to a student's view of her ability to complete learning objectives and to reach goals, and this characteristic may be particularly influential during tutorial dialogue for CS. This paper examines a corpus of effective human tutoring for CS to discover the extent to which considering self-efficacy as measured within a student pre-survey, coupled with dialogue and task events unfolding during tutoring, improves models that predict the student's self-reported frustration and learning gains after tutoring. The analysis reveals that particular patterns of student and tutor behavior are associated with different outcomes for students with low and high general self-efficacy. It is hoped that this line of research will enable tutoring systems for CS to tailor their tutorial interactions more effectively.

Keywords: Self-Efficacy, Tutorial Dialogue, Learning, Frustration

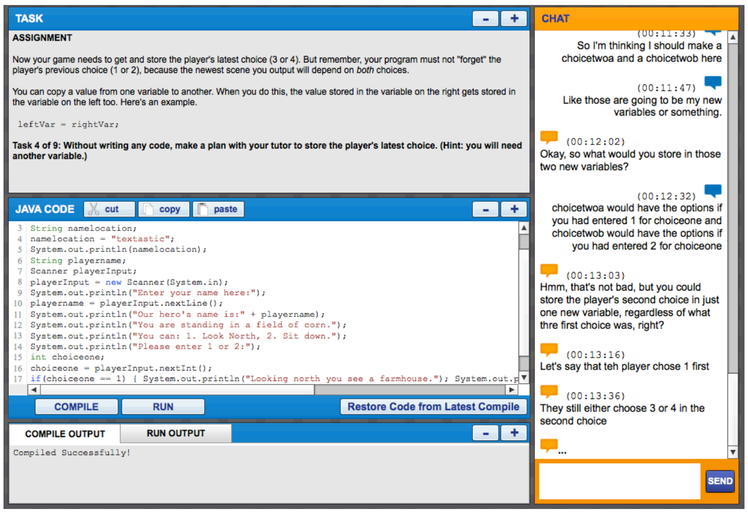
1 Introduction and Related Work

In terms of effectiveness, one-on-one tutoring has been shown to outperform conventional classroom settings [1, 12], but the mechanisms responsible have yet to be fully understood. Various studies have examined the interaction between students and tutors in terms of cognitive and affective outcomes [2], the adaptive presentation of instructional material [6], motivational strategies [8], and the exchange of rich natural language dialogue [9]. These modules have been implemented in various tutorial dialogue systems in the hopes of creating a tutorial dialogue system with effectiveness that rivals one-on-one human tutoring. Significant progress has been made in recent years as tutorial dialogue systems have been implemented and tutorial tactics tested in domains such as physics [5], electricity and electronics [7], and as is the focus of this paper, computer science [3].

It is generally acknowledged that students do not benefit equally from learning sessions with intelligent tutoring systems [12]. While tutoring has a negligible impact on some students, others thrive and accomplish significant learning. Some of the disparity can be explained by modeling the student's interest in the subject, memory capacity, and attentional abilities [6] but these differences do not capture all the variance. Therefore, predicting which tutorial interventions are most likely to support learning for particular students is a key open question for tutorial dialogue researchers. This paper examines that question by building models to predict the effectiveness of tutorial dialogue within a corpus of human-human tutoring for introductory CS. We report on separate models for students with high versus low self-efficacy, and the models show that learning and frustration are predicted by substantially different sets of dialogue indicators for students depending on their incoming self-efficacy.

2 Tutorial Dialogue Corpus

Our corpus consists of human-human computer-mediated tutorial dialogue for introductory computer science collected during the 2011-2012 academic year. This work is part of the larger JavaTutor project, which aims to create a tutorial dialogue system that learns its behavior from corpora with experienced human tutors. Students (*N*=66) and human tutors interacted through a web-based integrated development environment [10] in which they were presented learning tasks and textual dialogue (Fig. 1).



ASSIGNMENT

Now your game needs to get and store the player's latest choice (3 or 4). But remember, your program must not "forget" the player's previous choice (1 or 2), because the newest scene you output will depend on *both* choices.

You can copy a value from one variable to another. When you do this, the value stored in the variable on the right gets stored in the variable on the left too. Here's an example.

```
leftVar = rightVar;
```

Task 4 of 9: Without writing any code, make a plan with your tutor to store the player's latest choice. (Hint: you will need another variable.)

```
String namelocation;
namelocation = "textastic";
System.out.println(namelocation);
String playername;
Scanner playerInput;
playerInput = new Scanner(System.in);
System.out.println("Enter your name here:");
playername = playerInput.nextLine();
System.out.println("Our hero's name is:" + playername);
System.out.println("You are standing in a field of corn.");
System.out.println("You can: 1. Look North, 2. Sit down.");
System.out.println("Please enter 1 or 2:");
int choiceone;
choiceone = playerInput.nextInt();
if(choiceone == 1) { System.out.println("Looking north you see a farmhouse."); System.out.p
```

```
Compiled Successfully!
```

(00:11:33)
So I'm thinking I should make a choicetwoa and a choicetwob here

(00:11:47)
Like those are going to be my new variables or something.

(00:12:02)
Okay, so what would you store in those two new variables?

(00:12:32)
choicetwoa would have the options if you had entered 1 for choiceone and choicetwob would have the options if you had entered 2 for choiceone

(00:13:03)
Hmm, that's not bad, but you could store the player's second choice in just one new variable, regardless of what thre first choice was, right?

(00:13:16)
Let's say that teh player chose 1 first

(00:13:36)
They still either choose 3 or 4 in the second choice

...

Fig. 1. The textual tutorial dialogue interface for introductory Java programming

The participants were university students in the United States with an average age of 18.5 years (*stdev* = 1.5). Each student was paired with a tutor for a total of six sessions on different days, limited to forty minutes each session. There were a total of 15,819 task actions and dialogue moves that corresponded to the first lesson. The set of possible interaction events, including student dialogue messages, tutor dialogue messages, and task actions are shown in Table 1.

Before tutoring, a pre-survey was administered containing two questionnaires for self-efficacy: general and CS-specific. *General self-efficacy* is a measure of how students view their own ability to complete tasks and reach their goals. The higher a student's general self-efficacy is, the more a student believes in her ability to learn and grow academically. To measure student self-efficacy, a scale developed by Chen et al. was used [4]. *CS-specific self-efficacy* is the student's belief in her ability to learn the particular domain of CS. To measure student self-efficacy for CS, a scale developed by Wiebe et al. was used [13].

Table 1. List of task actions and dialogue moves

Task Action	Description
SESSIONPROGRESS	Tutor has moved the session forward to the next task
TUTORMSG	Tutor has sent a message to the student
STUDENTMSG	Student has sent a message to the tutor
CODE+	Student has written code that is closer in terms of edit distance to the correct solution* than the prior code
CODE-	Student has written code that is farther in terms of edit distance from the correct solution* than the prior code
CODE0	Student has written code that is neither closer nor farther from the correct solution* in terms of edit distance than the prior code (e.g. changing string literals)
COMPILEBEGIN	Student has clicked the compile button and is attempting to compile the current code
COMPILESUCCESS	Student's code had no syntax errors and compiled successfully
COMPILEERROR	Student's code had syntax errors and could not compile successfully
RUNCODE	Student has clicked the run button
RUNSUCCESS	Student has finished running the code with no errors
RUNTIMEERROR	Student has encountered an error while running the program
RUNSTOP	Student has halted the currently active program by clicking stop
INPUTSENT	Student has entered console input to test program

** A correct solution is any code that a tutor allowed to progress to the next task*

To measure learning gain for the lesson, an identical pre/post test was administered and normalized learning gain was calculated for each student during that lesson [10]. Finally, after each lesson, a post-session survey was administered to the students. This survey included questions used to determine the affective outcomes of the lesson, including student frustration and student engagement. Of particular interest for the current study is the student's self-reports of frustration and self-efficacy, each measured on a scale from 0 to 100.

3 Investigating Self-Efficacy

This study aims to identify differences in effective tutor interactions with students of high versus low self-efficacy. However, we were faced with multiple measures of self-efficacy with different theoretical properties. Our data do indeed reflect differences in these two measures; for example, CS-specific self-efficacy is slightly positively correlated with pretest score (indicating students who knew more did have slightly higher confidence), while general self-efficacy was slightly negatively correlated with pretest score, although this correlation was not statistically significant. In addition to different relationships with the incoming characteristic of pretest score, the self-efficacy measures behave differently in terms of their relationship with outcomes. As shown in Table 2, we see that General Self-Efficacy was significantly correlated with Normalized Learning Gain and Frustration but not with Engagement. However, CS-Specific Self-Efficacy is not significantly correlated with any of the session outcomes considered. Therefore, the remaining analyses proceed by considering General Self-Efficacy.

Table 2. Self-efficacy correlations with student outcomes

	General Self-Efficacy		**CS-Specific Self-Efficacy**	
	r	***p***	***r***	***p***
Pretest	-0.1795	0.1493	0.2569	0.0373
Normalized Learning	0.3406	0.0055*	0.2550	0.0403
Frustration	-0.3317	0.0051*	-0.1529	0.2203
Engagement	0.0559	0.6560	0.1389	0.2662

**p value that is significant after Bonferroni threshold of 0.00833*

4 Predictive Models

We proceed by splitting students based on their General Self-Efficacy scores (for the remainder of this paper simply referred to as *self-efficacy*). In order to explore difference in learning gain and frustration predictors between students of high and low self-efficacy, our analysis divides the 66 students into two mutually exclusive bins on which stepwise linear regression models are built. The high self-efficacy bin (N=30) contains the students whose general self-efficacy was above the median of 4 on a scale of 1 to 5. The low self-efficacy bin (N=36) are those that had a general self-efficacy equal to or below the median. We computed bigrams (pairs of adjacent interaction events) across all interaction events listed in Table 1, and these bigrams were provided to the models as predictors, with normalized learning gain and frustration as response variables. Initial feature selection was performed using model-averaging in JMP statistical software, which created regression models for all possible combinations of predictive variables [11]. The twenty most predictive variables were selected using the average coefficient estimate from models with one, two, or three predictive variables. Forward stepwise feature selection was then performed with leave-one-out cross-validated R^2 value as the selection criterion. Bayesian Information Criterion (BIC) was used to limit the number of predictive variables based on the BIC penalty for model complexity. For both learning gain and frustration, separate models were built for students of high and low self-efficacy.

4.1 Predicting Normalized Learning Gain

High Self-Efficacy Students. Table 3 shows the model of learning gain for high self-efficacy students. In this group, compiling code followed by compile errors is associated with lower learning gains. For all students, but perhaps markedly for students with high self-efficacy, these failed compile attempts may represent unproductive events. Another significant bigram is a tutor sending a message followed by the student running the code, which is also associated with negative learning. In our corpus, these bigrams often involve the tutor issuing directives rather than allowing the student to control the flow of the lesson.

This model also reveals bigrams that are associated with higher learning gains. The first bigram is the student running the code and then manually stopping the run. This move likely reflects the student realizing an issue in his code and stopping the run to correct it. The other two bigrams that are positively associated with learning are code quality improvement followed by the student sending a message to the tutor, and running the code followed by messaging the tutor. Taking the initiative to reflect or ask a follow-up question after making a code improvement or executing the code may be particularly helpful for these students. The example in Figure 2 illustrates an interaction displaying an increase in code quality followed by a student utterance, in which the student has updated code and is asking if it is correct, and then a compile attempt followed by an error. The second excerpt is the tutor prompting the student to compile, the student doing so and acknowledging, and then entering the necessary inputs.

Table 3. Model of learning gain for high self-efficacy students (R^2: 0.764; r: 0.874)

Normalized Learning Gain =	**Partial R^2**	**Model R^2**	***p***
-0.19 * COMPILEBEGIN_COMPILEERROR	0.308	0.308	<0.001
-0.10 * TUTORMSG_RUNCODE	0.202	0.510	0.004
0.20 * RUNCODE_RUNSTOP	0.159	0.669	0.003
0.10 * CODE+_STUDENTMSG	0.059	0.728	0.061
0.14 * RUNCODE_STUDENTMSG	0.036	0.764	0.072
0.60 (intercept)			<0.001
RMSE = 0.561 standard deviations **Leave-One-Out Cross-Validated R^2** = 0.633			

Excerpt #1
Student: [CODE+] does that work?
Tutor: try it
Student: [COMPILEBEGIN] [COMPILEERROR]

Excerpt #2
Tutor: ready?
Student: [RUNCODE] yes [INPUTSENT] [INPUTSENT] [INPUTSENT] [RUNSUCCESS]
Tutor: ok [SESSIONPROGRESS]

Fig. 2. Dialogue excerpts with two different high self-efficacy students

Low Self-Efficacy. Table 4 shows the model of learning gain for low self-efficacy students. This model has two bigrams associated with lower learning gains. The first is a tutor message to the student followed by a decrease in the quality of the code. It is possible that the student was not able to correctly operationalize the advice given by the tutor in these situations. The second bigram is a compile success followed by the tutor advancing the task. From examining our corpus, it is likely that these events could have resulted in the student feeling rushed because the next subtask was displayed before the student was able to execute the newly compiled code.

This model also reveals two bigrams associated with higher learning for students with low self-efficacy. The first is a student message followed by an increase in code quality. When students make dialogue contributions and then undertake productive task actions, this likely indicates successfully carrying out a localized plan. The second bigram is a code change that does not increase or decrease the quality of the code followed by another such code change. These superficial changes, such as correcting a misspelling in one of their string literals or editing a comment, may represent ongoing engagement in the task or taking the initiative to personalize the code. An example of an interaction with a low self-efficacy student can be seen in Excerpt 1 of Figure 3 in which a tutor has messaged the student with an unclear direction and the student introduces an error into the code. The second excerpt displays a tutor prompting the student to begin editing the code and the student making progress towards the correct solution and then changing the aesthetics of the code.

Table 4. Model of learning gain for low self-efficacy students (R^2: 0.420; r: 0.648)

Normalized Learning Gain =	**Partial R^2**	**Model R^2**	***p***
0.13 * STUDENTMSG_CODE+	0.142	0.142	0.042
-0.22 * TUTORMSG_CODE-	0.095	0.237	0.003
0.20 * CODE0_CODE0	0.110	0.347	0.015
-0.08 * COMPILESUCCESS_SESSIONPROGRESS	0.073	0.420	0.057
0.39 (intercept)			<0.001
RMSE = 0.751 standard deviations **Leave-One-Out Cross-Validated R^2** = 0.195			

Excerpt #1
Student: okay, so is it the same exact thing we did in lines 3 and 4?
Tutor: yes just line 3 for now
Student: [CODE-] [COMPILEBEGIN] [COMPILEERROR]

Excerpt #2
Tutor: now we will work on your program.
Student: ok [CODE+] [CODE0] [CODE0] [CODE0]
Tutor: good. ready?

Fig. 3. Dialogue excerpts from two different low self-efficacy students

4.2 Predicting Frustration

High Self-Efficacy. Table 5 shows the model of frustration for high self-efficacy students. This model identifies four bigrams as being associated with higher frustration for high self-efficacy students. The first bigram is a decrease in code quality while the code is running followed by the student stopping the running code. This code edit may have been made under the incorrect assumption that it would correct the problem observed during the run, revealing an important misunderstanding by the student. The second bigram is a compile success followed by an additional compile attempt. Clicking the compile button multiple times in succession may be a sign of disengagement or impatience with the compile process. The third bigram is the student running the code followed by the tutor sending a message. This may indicate a slightly higher level of tutor initiative than is ideal for high self-efficacy students. The final bigram is a student message being followed by another student message. In our corpus for high self-efficacy students, multiple messages being sent from the student before the tutor responded may reflect particular uncertainty from the student. An example of an interaction with a high self-efficacy student can be seen in Figure 4 in which a student has run the code and the tutor asks a question about a potential action that would introduce error into the code. The student makes the change to the code and then stops the current run in order to attempt another compile which fails.

Table 5. Model of frustration for high self-efficacy students (R^2: 0.721; r: 0.849)

Frustration Level =	**Partial R^2**	**Model R^2**	***p***
7.75 * CODE-_RUNSTOP	0.345	0.345	<0.001
4.79 * COMPILESUCCESS_COMPILEBEGIN	0.167	0.512	0.010
5.81 * RUNCODE_TUTORMSG	0.138	0.650	0.001
3.90 * STUDENTMSG_STUDENTMSG	0.071	0.721	0.018
11.29 (intercept)			<0.001
RMSE = 0.441 standard deviations **Leave-One-Out Cross-Validated R^2** = 0.222			

Student: [RUNCODE]
Tutor: good. what if we commented out line 6?
Student: we would get an error
Tutor: java would flag what lines as errors?
Student: 7,8,10,11
Tutor: 10 and 11 yes 7 and 8 no. Try it. First comment out line 6
Student: [CODE-] [RUNSTOP] [COMPILEBEGIN] [COMPILEERROR]

Fig. 4. Dialogue excerpt from high self-efficacy student

Low Self-Efficacy. Table 6 shows the model predicting frustration for low self-efficacy students. This model identified six bigrams as being associated with higher frustration for low self-efficacy students. The first bigram is a compile success followed by the tutor advancing to the next task. This bigram was also identified as being associated with lower learning gain, and the potential to make the student feel rushed could also have led to increased frustration. The second bigram is the student running the code and the tutor advancing the task before the run had completed. Like the previous bigram, this move very likely caused the student to feel rushed, increasing frustration. The third bigram was the code decreasing in quality during a run, followed by the student stopping the current run, which was also present in the model for high self-efficacy students. Editing the code while it is running indicates an attempt to repair a bug, but this attempt is not productive, possibly leading to frustration. The fourth bigram is the student entering input into the running code and then sending the tutor a message. This is often a pattern observed in the corpus when the student is expressing confusion about what is happening in the running code to the tutor. The fifth bigram is two events of code quality decreasing. Finally, a compile success followed by a tutor messaging the student was also associated with increased frustration. It is possible that the positive feedback or elaboration provided by the tutor at these times was not optimal for supporting the student's affective state in terms of minimizing frustration.

Table 6. Model of frustration for low self-efficacy students (R^2: 0.883; r: 0.940)

Frustration Level =	**Partial R^2**	**Model R^2**	***p***
-1.93 * COMPILESUCCESS_STUDENTMSG	0.333	0.333	0.464
10.37 * COMPILESUCCESS_SESSIONPROGRESS	0.143	0.476	<0.001
-6.63 * TUTORMSG_COMPILEBEGIN	0.172	0.648	<0.001
1.62 * RUNCODE_SESSIONPROGRESS	0.044	0.692	0.149
10.51 * CODE-_RUNSTOP	0.040	0.732	<0.001
3.36 * INPUTSENT_STUDENTMSG	0.065	0.797	0.065
-4.89 * SESSIONPROGRESS_COMPILEBEGIN	0.038	0.835	0.014
2.70 * CODE-_CODE-	0.020	0.855	0.045
-4.08 * INPUTSENT_RUNSUCCESS	0.014	0.869	0.066
2.83 * COMPILESUCCESS_TUTORMSG	0.014	0.883	0.099
11.84 (intercept)			<0.001
RMSE = 0.466 standard deviations **Leave-One-Out Cross-Validated R^2** = 0.383			

The model also reveals four bigrams that are associated with lower frustration for low self-efficacy students. The first bigram is a compile success followed by the student messaging the tutor, and may indicate student reflection or elaboration upon the success. The second bigram is a tutor messaging the student followed by the student compiling the program. A tutor's direct support that leads to compiling may precede successes, decreasing frustration. The third bigram is the tutor progressing to the next task and then the student attempting to compile. This could be indicative of the student taking a proactive approach, testing the prior tasks' code and reviewing the areas that will need improvement for the current task. The final bigram is the student sending input to the running program followed by the run completing successfully. This bigram indicates a full successful test. An example of an interaction with a low self-efficacy student can be seen in Figure 5 in which the student asks for the tutor's approval for compilation, receives it, successfully compiles, and then informs the tutor of the success.

Student: Should I compile?
Tutor: Please do
Student: [COMPILEBEGIN] [COMPILESUCCESS] Done.
Tutor: Alright, we'll move on to the next step. [SESSIONPROGRESS]

Fig. 5. Dialogue excerpt of low self-efficacy student

Discussion. As the models display, high and low self-efficacy students may have substantially different tutorial needs than one another. In models to predict learning gain and frustration, while some predictors were held in common across the self-efficacy groups, the majority of tutorial dialogue indicators differed between the models. Overall, the results suggest that for low self-efficacy students, higher tutor involvement is called for. These students benefitted, for example, from tutor messages just before compilation, which was associated with decreased frustration. In the case of high self-efficacy students, several types of tutor intervention were associated with negative impacts. Following running of the student's code with an immediate tutor message was associated with higher frustration for these students, while a tutor message followed by running the code was associated with decreased learning gains. High self-efficacy students may benefit from more autonomy and space to explore the meaning of their results before receiving tutor interventions.

5 Conclusion and Future Work

Student self-efficacy may have deep and far-reaching implications for computer science learning and frustration levels after engaging in learning tasks. It was observed that general self-efficacy was more highly correlated with learning gain and frustration than the CS-specific self-efficacy. This paper has examined predictive models of learning and frustration for students with high and low general self-efficacy, finding that different interactions were predictive of these outcomes for the two groups. These models imply that tutor proactivity was well received by the low self-efficacy students, correlating with increased learning gains and decreased frustration, while it may have hindered the high self-efficacy students, lowering learning gains and increasing frustration.

Individual adaptation is a crucial goal for computer science tutoring systems, which hold great potential to support broader populations of students in learning computer science. As a community we must work to enable future automated tutoring systems to be customized to students' individual needs and preferences, increasing both their affective outcomes and learning.

Acknowledgements

This work is supported in part by the North Carolina State University Department of Computer Science and the National Science Foundation through Grant DRL-1007962. Any opinions, findings, conclusions, or recommendations expressed in this report are those of the participants, and do not necessarily represent the official views, opinions, or policy of the National Science Foundation.

References

1. Bloom, B. The 2 Sigma problem: the search for methods of group instruction as effective as one-to-one tutoring. *Educational Researcher*, 13(6): 4-16 (1984).
2. Boyer, K., Phillips, R., Wallis, M., Vouk, M., & Lester, J. Balancing cognitive and motivational scaffolding in tutorial dialogue. In *Proceedings of ITS*. 239-249 (2008).
3. Chen, L., Di Eugenio, B., Fossati, D., Ohlsson, S. & Cosejo, D. Exploring effective dialogue act sequences in one-on-one computer science tutoring dialogues. In *Proceedings of the Sixth Workshop on Innovative Use of NLP for Building Educational Applications*. 65-75 (2011).
4. Chen, G., Gully, S. M., & Eden, D. Validation of a new general self-efficacy scale. *Organizational research methods*, 4(1): 62-83 (2001).
5. Chi. M., VanLehn, K. & Litman, D. Do micro-level tutorial decisions matter: applying reinforcement learning to induce pedagogical tutorial tactics. In *Proceedings of ITS*. 224-234 (2010).
6. D'Mello, S.K., Williams, C., Hays, P. & Olney, A. Individual differences as predictors of learning and engagement. In *Proceedings of the Annual Meeting of the Cognitive Science*. 308-313 (2009).
7. Dzikovska, M., Steinhauser, N., Moore, J.D., Campbell, G.E., Harrison, K.M. & Taylor, L.S. Content, social, and metacognitive statements: an empirical study comparing human-human and human-computer tutorial dialogue. In *Proceedings of the European Conference on Technology Enhanced Learning*. 93-108 (2010).
8. Lepper, M.R., Woolverton, M., Mumme, D.L. & Gurtner, J.L. Motivational techniques of expert human tutors: lessons for the design of computer-based tutors. *Computers as Cognitive Tools*. 75-105 (1993).
9. Litman, D., Moore, J.D., Dzikovska, M. & Farrow, E. Using natural language processing to analyze tutorial dialogue corpora across domains and modalities. In *Proceedings of AIED*. 149-156 (2009).
10. Mitchell, C., Ha, E., Boyer, K. & Lester, J. Learner characteristics and dialogue: recognizing effective and student-adaptive tutorial strategies. *International Journal of Learning Technology (IJLT)*, 8(4): 382-403 (2013).
11. Symonds, M.R.E. and Moussalli, A. A brief guide to model selection, multimodel inference and model averaging in behavioral ecology using Akaike's information criterion. *Behavioral Ecology and Sociobiology*, 65(1): 13-21 (2010).
12. VanLehn, K., Graesser, A. C., Jackson, G. T., Jordan, P., Olney, A., & Rose, C. P. When are tutorial dialogues more effective than reading? *Cognitive Science*, 31(1): 3-62 (2007).
13. Wiebe, E. N., Williams, L., Yang, K. & Carol Miller, C. Computer science attitude survey. (Report No.: NCSU CSC TR-2003-1) Dept. of Computer Science, NC State University, Raleigh, NC. (2003).