

Evaluating Intelligent Tutoring Systems with Learning Curves

Brent Martin and Antonija Mitrovic

Intelligent Computer Tutoring Group
Department of Computer Science, University of Canterbury
Private Bag 4800, Christchurch, New Zealand
{brent,tanja}@cosc.canterbury.ac.nz

Abstract. The evaluation of Intelligent Tutoring Systems, like any adaptive system, can have its difficulties. In this paper we discuss the evaluation of an extension to an existing system that uses Constraint-Based Modelling (CBM). CBM is a student modelling technique that is rapidly maturing, and is suited to complex, open-ended domains. A problem with complex domain models is their large size, necessitating a comprehensive problem set in order to provide sufficient exercises for extended learning sessions. We have addressed this issue by developing an algorithm that automatically generates new problems directly from the domain knowledge base. However, evaluation of this approach was complicated by the need for a lengthy (and therefore uncontrolled) study as well as other unavoidable differences between the control and experimental systems. This paper presents the evaluation and discusses those issues, and the way in which we used learning curves as a tool for comparing disparate learning systems.

1 Introduction

Constraint-Based Modelling (CBM) [7] is an effective approach that simplifies the building of domain and student models in Intelligent Tutoring Systems (ITS). We have used CBM to develop SQL-Tutor [4], an ITS for teaching the SQL database language. SQL-Tutor supports learning in three ways: by presenting feedback when students submit their answers, by controlling problem difficulty, and by providing scaffolding information. Students have shown significant gains in learning after as little as two hours of exposure to this system [5].

SQL-Tutor contains a list of problems, from which one is selected that best fits the student's current knowledge state. In extended sessions with the tutor the system may run out of suitable problems. We have attempted to overcome this by developing a problem generator that uses the domain model to build new problems that fit the student model, and a problem selection algorithm that is tied more closely to the student model.

We evaluated the new algorithm against the existing SQL-Tutor system in a classroom setting, but had difficulty because there were other differences in the system besides what we were testing. Also, the evaluation was over an extended period in an uncontrolled environment, which made comparisons in student performance difficult. We were therefore unlikely to see any difference in outcome using purely subject pre- and post-testing.

In this paper we briefly introduce SQL-Tutor and the extensions to it. We then describe the experiment and its evaluation. We discuss the use of learning curves to analyse the performance of the two systems, and introduce a new measure, that of the initial learning rate. We compare this new measure to the usual ones of slope and intercept, and discuss why we think it is valid. Finally, we conclude why we think the initial learning rate may be a better measure for determining how much learning went on while students used each system.

2 SQL-Tutor

SQL-Tutor [4] teaches the SQL database query language to second and third year students at the University of Canterbury, using Constraint-Based Modelling (CBM). This approach models the domain as a set of state constraints, of the form:

If <relevance condition> is true for the student's solution,
THEN <satisfaction condition> must also be true

The relevance condition of each constraint is used to test whether the student's solution is in a pedagogically significant state. If so, the satisfaction condition is checked, which tests for valid constructs (syntactic constraints) or compares the solution to an ideal solution (semantic constraints). If a constraint succeeds, no action is taken; if it fails, the student has made a mistake and appropriate feedback is given. CBM has advantages over other approaches such as model tracing [1] in that the model need not be complete and correct in order to function. Further, it is well suited to domains where the number of alternative solutions is large or the domain is open-ended.

Ohlsson [7] does not impose any restrictions upon how constraints are encoded, and/or implemented. In SQL-Tutor we initially represented each constraint by a LISP fragment, supported by domain-specific LISP functions. In later versions we have used a pattern-matching algorithm designed for this purpose [3], for example:

```
(147
"You have used some names in the WHERE clause that are not from
this database."
(match SS WHERE (?* (^name ?n) ?*))
(or (test SS (^valid-table (?n ?t))
(test SS (^attribute-p (?n ?a ?t))))
"WHERE")
```

The above constraint checks for valid table and attribute names in the WHERE clause of the student's SQL statement. It is relevant if any names exist in the WHERE clause, and satisfied if each of these is either a valid table name or a valid attribute name.

The constraint language makes all of the logic for determining whether or not the constraint is satisfied transparent to the system, since it consists only of pattern matching and logical combination. Functions, such as “^valid-table” in the above example, are simply macros defined in the same language. We have used this property to develop a problem solver that can generate correct solutions from student

attempts by extracting (and unifying) the valid match patterns from each satisfied constraint and the *invalid* patterns from violated constraints. The algorithm then corrects the invalid fragments by unifying them against matched patterns of the *ideal* solution, and then combines the resulting solution fragments.

Unlike repair theory [8], we make no claim that this algorithm is modelling human behaviour. However, it has the advantage that a failed constraint means that the construct involved is *definitely wrong*: we do not need to try to infer where the error lies, so our algorithm does not suffer from computational explosion. For further details on this algorithm and the constraint language refer to [3].

3 Generating New Problems

In the original version of SQL-Tutor the next problem is chosen based on difficulty, plus the concept the student is currently having the most trouble with. The constraint set is first searched for the constraint that is being violated most often. Then the system identifies problems for which this constraint is relevant. Finally, it chooses the problem from this set that best matches the student's current proficiency level. However, there is no guarantee that an untried problem exists that matches the student model: there may be no problems for the target constraint, or the only problems available may be of unsuitable difficulty. Further, since the constraint set is large (over 650 constraints), many problems are needed merely to cover the domain. Ideally there should be many problems per constraint, in various combinations. In SQL-Tutor there is an average of three problems per constraint and only around half of the constraint set is covered. A consequence of this is that the number of new constraints being presented to the student tapers off as the system runs out of problems.

The obvious way to address this limitation is to add more problems. However, this is not an easy task. There are over 650 constraints, and it is difficult to invent problems that are guaranteed to cover all constraints in sufficient combinations that there are enough problems at a large spread of difficulty levels. To overcome this we have developed an algorithm that generates new problems from the constraint set. It uses the constraint-based problem solver described in Section 2 to create novel SQL statements, using an individual constraint (or, possibly, a set of compatible constraints) to provide a set of SQL fragments as a starting point. These are then "grown" into a full SQL statement by repeatedly splicing them together and unifying them against the syntactic constraint set until no constraints are violated. This new SQL statement forms the ideal solution for a new problem. The human author need only convert the ideal solution into a natural language problem statement, ready for presentation to the student.

We used the problem generation algorithm create a single problem per constraint, giving around 800 potential ideal solutions (note that the experimental version had an extra 250 constraints added). We then chose the best of these and converted them into natural language problem statements. On completion we had a new problem set of 200 problems, which took less than a day of human effort to build. Furthermore, when we plotted the number of new constraints applied per problem presented to a student, the point at which new problems failed to introduce any new concepts (i.e. previously unseen constraints) rose from 40 problems to 60, indicating that the new problem set increased the length of time that a student could fruitfully engage with the system.

The experimental system also used a different problem selection mechanism. In the control system, a new problem is selected by finding the constraint the student is currently violating most often and selecting the problem whose (static) difficulty rating is closest to the student's current proficiency level. In the experimental system we calculated the difficulty of each problem dynamically by computing its static difficulty from the number of constraints (and their complexity) relevant to it, and then added further difficulty for relevant constraints that the student is currently violating, and more still for constraints that the student has not yet encountered. Thus each problem is compared to the student's current knowledge. Once difficulties have been calculated for all problems, the one that is closest to the student's current proficiency is selected.

4 Evaluation

The motivation for Problem Generation was to reduce the effort involved in building tutoring systems by automating one of the more time-consuming functions: writing the problem set. Three criteria must be met to achieve this goal: the algorithm must work (i.e. it must generate new problems); it must require (substantially) less human involvement than traditional problem authoring; and the problems produced must be shown to facilitate learning to at least the same degree as human-authored problems. The first two were confirmed during the building of the evaluation system: the algorithm successfully generated problems, and the time taken to author the problem set was much less than would have been required for human authoring alone.

Additionally, if the method works, it should be possible to generate large problem sets, which will have the benefit of greater choice when trying to fit a problem to the user's current student model. We might therefore expect that given a suitable problem selection strategy, a system using the generated problem set would lead to faster learning than the current human-authored set, because we are better able to fit the problem to the student.

SQL-TUTOR was modified for this purpose and evaluated for a six-week period. The subjects were second year University students studying databases. The students were broken into three groups. The first used the current version of SQL-TUTOR, i.e. with human-authored problems. The second group used a version with problems generated using the algorithm described. The third group used a version containing other research (student model visualisation) that was not relevant to this study. Before using the system, each student sat a pre-test to determine their existing knowledge and skill in writing SQL queries. They were then free to use the system as little or as often as they liked over a six week period. Each student was randomly assigned a "mode" that determined which version of the system they would use. At the conclusion of the evaluation they sat a post-test.

When the study commenced, 88 students had signed up and sat the pre-test, giving sample sizes of around 30 per group. During the evaluation this further increased as new students requested access to the system. At the conclusion of the study some students who signed up had not used the system to any significant degree. The final groups used for analysis numbered 24 (control) and 26 (experimental) students each. The length of time each student used the system varied greatly from not using it at all

to working for several hours, with an average of 2½ hours. Consequently, the number of problems solved also varied widely from zero to 98, with an average of 25.

There are several ways we can measure the performance of the system. First, we can measure the means of the pre-test and post-test to determine whether or not the systems had differing effects on test performance. Note, however, that with such an open evaluation as this it is dangerous to assume that differences are due to the system, since use of the system may represent only a portion of the effort the student spent learning SQL. Nevertheless, it is important to analyse the *pre-test* scores to determine whether the study groups are comparable samples of the population. This was found to be the case. There was similarly no significant difference in post-test scores, as we might expect.

Second, we can plot the reduction in error rates as the student practices on each constraint, or the “learning curve”. Each student’s performance when measured this way should lead to an exponential curve or so-called “Power law” [2, 6], which is typical when each underlying object being measured (in this case a constraint) represents a concept being learned. The steepness of this curve is an indication of the speed with which the student, on average, is learning new constraints. Since each constraint represents a specific concept in the domain, this is an indication of how quickly the student is learning the domain. We can then compare this learning rate between the two groups.

Finally, we can look at how difficult the students found the problems. This is necessary to ensure that the newly generated problems did not negatively impact problem difficulty (either by being too easy or too hard). There are several ways we can do this. First, we can measure how many attempts the student took on average to solve a problem and compare the means for the control and test groups. Second, students were permitted to abort the current problem and were asked to cite one of three reasons: it was too easy, it was too hard or they wanted to try a problem of a different type. If the proportion of problems aborted rises, or the ratio of “too hard” to “too easy” problems is further from 1:1 than the control group, we might conclude that problem difficulty has been adversely affected.

In this study, we measured all of the above. We used the software package SPSS to compare means and estimate power and effect size, and Microsoft Excel to fit power curves. We measured problem difficulty both subjectively and objectively: we obtained subjective results by logging when students aborted a problem and recording their reason. Any significant difference in the ratio of “too easy” to “too hard” responses would suggest we have adversely affected problem difficulty. Further, the percentage of problems aborted should not rise significantly. Next, we measured the number of attempts taken to solve each problem, and the time taken on each attempt. There was no significant difference in any of these comparisons. In other words, the students appeared to have found the level of difficulty of each problem about the same for both systems.

4.1 Learning Curves

Since the general tests failed to show any difference between the two groups, we turned to learning curves as a means of evaluating more closely what was occurring *while the system was being used*. We observed the learning rate for each group by plotting the proportion of constraints that are violated for the n th problem for which

this constraint is relevant. This value is obtained by determining for each constraint whether it is correctly or incorrectly applied to the n th problem for which it is relevant. A constraint is correctly applied if it is relevant at any time during solving of this problem, and is *always* satisfied for the duration of this problem. Constraints that are relevant but are violated one or more times during solving of this problem are labelled erroneous. The value plotted is the proportion of all constraints relevant to the n th problem that are erroneous.

If the unit being measured (constraints in this case) is a valid abstraction of what is being learned, we expect to see a “power curve”. We fitted a power curve to each plot, giving an equation for the curve. Note that as the curve progresses learning behaviour becomes swamped by random erroneous behaviour such as slips, so the plot stops trending along the power curve and levels out at the level of random mistakes. This is exacerbated by the fact that the number of constraints being considered reduces as n increases, because many constraints are only relevant to a small number of problems. We therefore use only the initial part of the curve to calculate the learning rate. Fig. 1 shows such plots, where each line is the learning curve for the entire group on average, i.e. the proportion of constraints that are relevant to the first problem that are incorrectly applied *by any student in the group*. The cut-off was chosen at $n=5$, which is the point at which the power curve fit for both groups is maximal. Note that learning curves are also exhibited when the data is plotted for a single student, although there is large variance between faster and slower learners, and the quality of the curves is often lower due to the small amount of data available. Similarly, plotting learning curves on a per-constraint basis (averaged over all students) gives power laws where the slope indicates the ease with which this constraint is learned, which is an indication of the quality of the constraint. For example, constraints that span more than one concept will produce a poor curve.

Both plots exhibit a very good fit to a power curve. The equations give us the Y intercept and slope for a log-log graph of constraint performance. In this case the experimental group had a Y intercept that was around *twice* that for the control group,

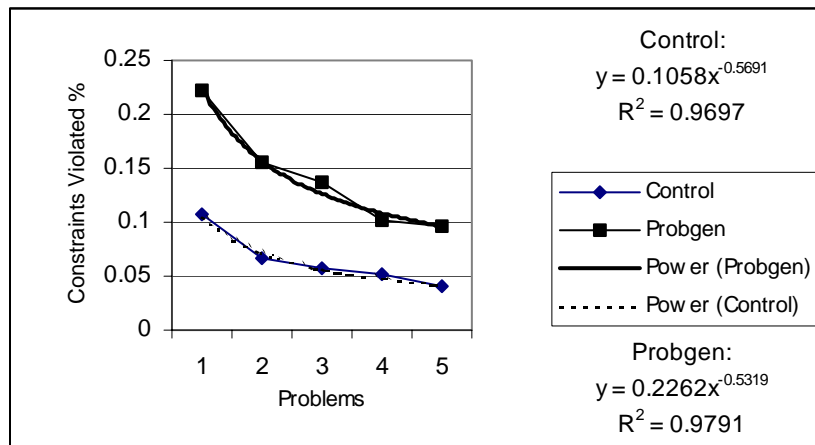


Fig. 1. Learning Performance

but a lightly lower slope (0.53 versus 0.57). Fig. 2 shows log-log plots for the same data.

4.2 Learning Curve Slope

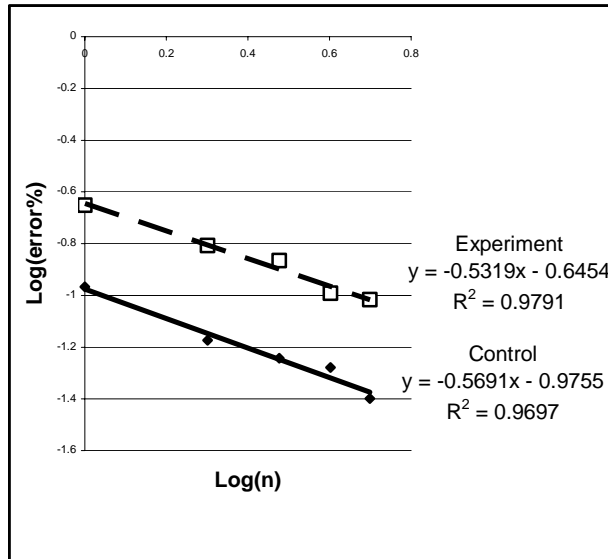


Fig. 2. Log-Log plots for the two groups.

The slope of a learning curve represents the *learning rate*, i.e. the rate at which the student decreased their errors with practice. There are several reasons why this might differ between the two groups: the students may have different average learning abilities; the constraints may represent domain concepts to a greater or lesser degree; the concepts may be introduced at more or less opportune times (and thus the students may be more or less receptive to learning the constraints at

that time).

The first difference (learner ability) was eliminated because the average pre-test scores were not significantly different for the two groups. Differences in the constraint sets were a possibility because the experimental system was a rewritten version of the control system using the new constraint representation: although the constraint set for the experimental group was based on that for the control group, a large number of modifications had been made, including the rewriting of many constraints that were always relevant, such that they were now only relevant in appropriate situations. New constraints were also added.

We tested the effect of these changes by recalculating the curves in two ways. First, we removed the data for all constraints that were trivially true (which occurred only in the control system's constraint set) and replotted the learning curve for the control group. Second, we took the raw student solutions from logs for the control group and evaluated them using the new constraint set. This latter method is a trade-off: it gives us the student's performance based on evaluation by the new constraints, but where the student still received feedback from the old constraint set.

In both cases the slope varied according to which method we used. Table 1 lists the results. The learning curve slope thus appears to be very sensitive to how the student's performance is being measured, and hence is not a suitable measure for comparing disparate systems.

Table 1. Learning curve slope for different constraint sets

Measuring method (Control Group)	Control Slope	Experiment Slope
Original constraint set	0.57	0.53
Exclude trivially true	0.31	0.53
Experimental constraint set	0.44	0.53

4.3 Y Intercept and Initial Learning Rate

The other measure of a learning curve is the Y intercept, which gives us the initial error rate. This alone is not a useful measure because it only indicates the student's initial performance, but does not show any effects of learning. However, the *slope* at $X=1$ (equal to the Y intercept multiplied by the log-log slope) does take learning into account: it indicates the raw improvement in performance achieved by a student between when they are first exposed to a constraint and when they have received feedback on it once. We therefore hypothesised that this is a better measurement of performance between disparate systems because it takes into account both the student's learning rate and the amount of unlearned material they are being exposed to as a percentage of the original constraint set. We expected that this calculation would be relatively invariant to changes in the constraint set, because it normalises out differences in the way the student's learning performance is measured.

Table 2 lists the results for the three ways that the students' performance was measured, i.e. using the original constraint, the same constraint set with trivial constraints excluded, and the new (experimental) constraint set. They show that, although the initial learning rate does differ depending on the constraint set used, the difference is much smaller than when the log-log slope is considered.

These results show that the slope at $X=1$, or initial learning rate, is almost twice as high for the experimental group. We checked for statistical significance by plotting learning curves for each individual student and comparing the means of the initial learning rate for the two groups. The difference was statistically significant at $\alpha=0.05$ ($p=0.01$). This suggests that the raw amount learned by the experimental group was higher than for the control group, which we attributed to either, or both, the increased number and range of problems available and an improved problem selection mechanism. Both of these could lead to a larger number of unknown constraints being presented to the student that were within their zone of proximal development [9]. Thus whereas the new problem set and selection method might not increase the student's learning rate, nevertheless it engages them in a larger volume of learning and therefore reduces the time taken to master the material.

Table 2. Initial slope for different constraint sets

Measuring method (Control Group)	Control Init. Slope	Exp. Init. Slope
Original constraint set	0.06	0.12
Exclude trivially true	0.05	0.12
Experimental constraint set	0.08	0.12

5 Discussion

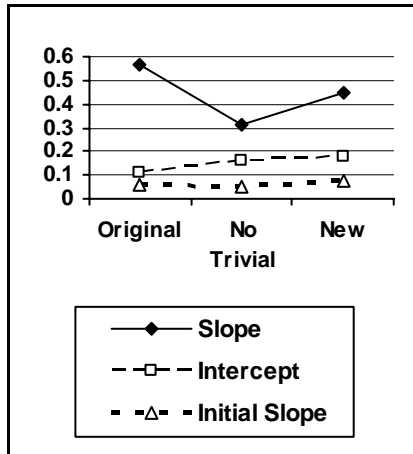


Fig. 3. Learning curve statistics.

only 10%, and is thus learning a greater volume of material at once. In both cases the ease with which the student learns the material is the same, but the amount they are learning varies widely. On the other hand, the Y intercept gives us the amount of learning being undertaken (i.e. what percentage of the presented material the student is trying to learn) but not the rate. However, the *initial* learning rate (slope at X=1) combines both the size of the learning task and the student's learning performance.

This new measure appears to be relatively invariant to the way student performance is measured, the issue in this case being differences in the constraint set. Fig. 3 plots how the measures of slope, Y intercept and initial slope vary with the constraint set used. Of the three, the initial slope varies the least. This is intuitively expected, and is illustrated in Fig. 4. These three curves are raw data for the control group, the same data multiplied by two, and again with a constant (0.5) added. The first two curves

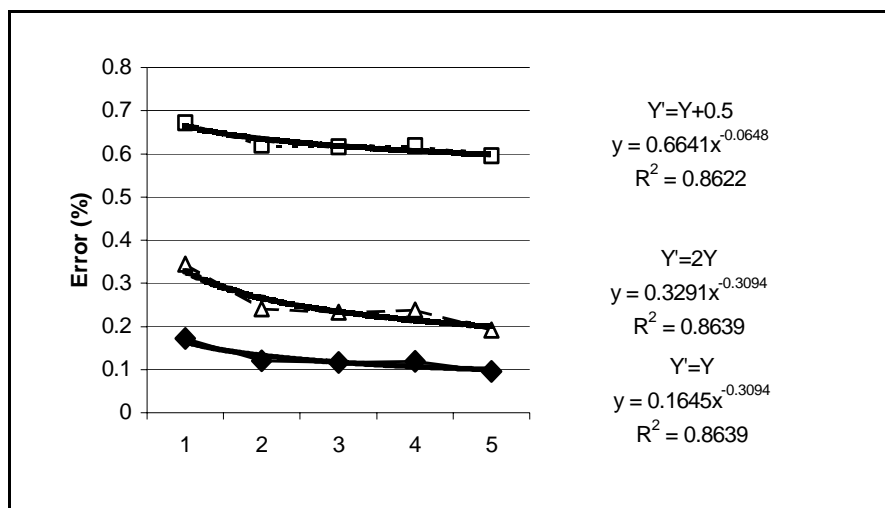


Fig. 4. Variations in power curve slope from artificial manipulation

have the same slope, even though the proportion of constraints being learned is twice as high for the second curve. However, the *initial* slope for the two curves has a ratio of 2:1, representing this effect. In contrast, the third curve is the same data with a constant amount added, which might represent a constraint set that includes trivially satisfied constraints. In this case the slope of the modified curve is dramatically lower (0.06 versus 0.3), even though the student's behaviour is unchanged. In contrast the initial learning rate is only slightly lower (0.43 versus 0.5).

6 Conclusions

This paper identified the problem of measuring student performance with intelligent tutoring systems when the systems being compared do not measure student performance in the same way. We showed that learning curves might still be used to compare such systems, but that the traditional measure of slope is not suitable because it varies with the method used to measure performance. We suggested a new statistic, that of the initial learning rate, which is produced by calculating the slope at $X=1$. We argued that this measure encompasses both the learning rate and the size of the learning task, and hence tells us more about the difference in performance between the two systems, because modifications to some aspects of a learning system (such as the problem set and the problem selection method) may alter the amount of material a student is learning at any time. We also showed that initial learning slope appears to be relatively robust against differences in the method used to measure the student's performance because it is normalised with respect to the student's initial performance.

Acknowledgement

This research was supported by the University of Canterbury grant U6430.

References

1. Anderson, J.R., Corbett, A.T., Koedinger, K.R., and Pelletier, R., *Cognitive Tutors: Lessons Learned*. Journal of the Learning Sciences, 1995. 4(2): pp. 167-207.
2. Anderson, J.R. and Lebiere, C., *The atomic components of thought*. 1998, MahWah, NJ: Lawrence Erlbaum Associates.
3. Martin, B. and Mitrovic, A. *Tailoring Feedback by Correcting Student Answers*. in *Fifth Int. Conf. on Intelligent Tutoring Systems*. 2000. Montreal: Springer. pp. 383-392.
4. Mitrovic, A. *Experiences in Implementing Constraint-Based Modeling in SQL-Tutor*. in *4th Int. Conf. on Intelligent Tutoring Systems*. 1998. San Antonio, Texas: Springer. pp. 414-423.
5. Mitrovic, A. and Ohlsson, S., *Evaluation of a Constraint-Based Tutor for a Database Language*. Int. J. Artificial Intelligence in Education, 1999. 10: pp. 238-256.
6. Newell, A. and Rosenbloom, P.S., *Mechanisms of skill acquisition and the law of practice*, in *Cognitive skills and their acquisition*, J.R. Anderson (ed.) 1981, Lawrence Erlbaum Associates: Hillsdale, NJ. pp. 1-56.
7. Ohlsson, S. *Constraint-Based Student Modeling*. in *NATO Advanced Research Workshop on Student Modelling*. 1991. Ste. Adele, Quebec, Canada: Springer-Verlag. pp. 167-189.
8. VanLehn, K., *On the Representation of Procedures in Repair Theory*, in *The Development of Mathematical Thinking*, H.P. Ginsburg (ed). 1983, Academic Press: New York. pp. 201-252.
9. Vygotsky, L.S., *Mind in society: The development of higher psychological processes*. 1978, Cambridge, MA: Harvard University Press.