An evolutionary approach to guiding students in an educational game

Elizabeth Sklar **Jordan Pollack**
DEMO Lab
Department of Computer Science
Brandeis University
Waltham, MA 02454-9110 USA
sklar,pollack@cs.brandeis.edu

Abstract

We describe an evolutionary approach to selecting content for educational games in a web-based learning community. Our approach offers an alternative to methods typically used in educational domains, with the goal of combining the curricular structure of an engineered application along with the flexibility of a learner-centered setting. Our method operates in a real-time environment, so performance requirements differ from those of an off-line implementation. We tested our method during a pilot study involving fourth and fifth grade students at a public primary school. This paper details our approach and presents results from the pilot study.

1. Introduction

Both computer games and educational software provide an interactive medium with which humans can explore a domain. In the case of computer games, the domain might be outer space or the wild west or a fantasy land; with educational software, the domain might be arithmetic or geography or spelling. In either case, the purpose of the software system is to guide a user through the domain in a methodical way, while exposing him to as much of the domain as possible without losing the user's interest and, especially in the case of educational software, while promoting learning.

In computer games, players are explicitly led from one "level" to the next and they are generally required to complete a level before being allowed to move ahead. Typically, the setting is competitive, in order to entice the player. Although this format has proven to be highly motivating for both children and adults alike, the method does not always provide an atmosphere dedicated to learning, nor is the progression customized to the needs of individual learners.

In educational software, users are often characterized according to their experience with the system and this information is stored in individualized *student models* (Greer and McCalla, 1994). Numerous and varied approaches are taken for student modeling, such as Bayesian techniques (VanLehn et al., 1998), reinforcement learning (Beck, 1997) and case-based reasoning (Shiri-A. et al., 1998). Systems use the student model as a guide for finding an appropriate portion of the domain with which to challenge the student. The student is then conducted through the domain, following a series of pre-defined paths of increasing difficulty and/or complexity. The student model is updated as the student progresses.

More recent trends in educational software move away from fixed, pre-programmed and/or pre-leveled environments and towards *constructivist*[1] environments where students are able to explore ideas for themselves without having to stick to fixed curricula, and students of all abilities are provided with opportunities to learn (Resnick, 1997, Papert, 1993). In a classroom setting, this notion has been described by Forrester (1992) as *learner-centered learning*:

> A teacher is no longer a dispenser of knowledge addressed to students as passive receptors. Instead, where small teams of students explore and work together and help one another, a "teacher" becomes a colleague and participating learner. Teachers set directions and introduce opportunities. Teachers act as guides. [p.11]

The same ideas apply to educational software, where the software system acts as the teacher. The system should be adaptive and participate in the learning, guiding users (students) through educational domains and adjusting as users advance.

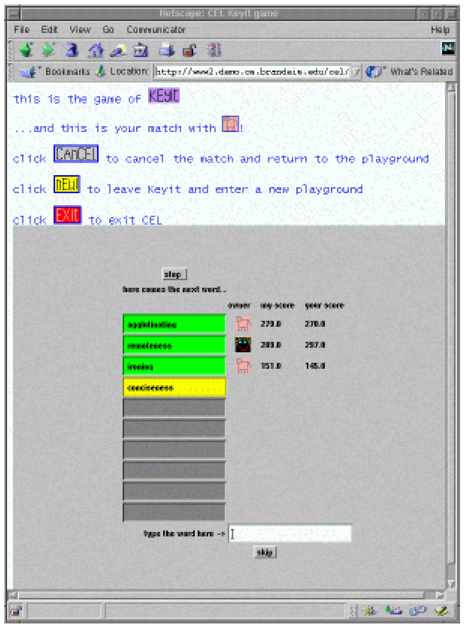
One paradigm that has been gaining popularity is the educational MUD[2]. Originating in the late 1960's as analog text-based role-playing adventure games, the MUDs have recently been introduced in educational applications (Fanderclai, 1995, Gordon and Hall, 1998). Two notable examples of educational MUDs implemented on the Internet are *Pueblo* (Walters and Hughes, 1994) and *MOOSE Crossing* (Bruckman, 1997). Both promote visualization and creative writing skills and have been

[1] This term is attributed to Jean Piaget.
[2] Multi-User Dungeon/Dimension/Domain

shown to draw otherwise uninterested children into literacy activities.

In these environments, the participants communicate through the language of the MUD, but otherwise the system itself does not guide the learning experience. Human teachers (or on-line MUD guides) motivate users to stay "on task". While these settings benefit learners by providing ultimate flexibility, we posit that they are only applicable to a limited number of situations, where the curriculum is highly open-ended, and so cannot completely replace the curricular structure offered by domain-engineered applications.

We use an evolutionary approach to guide students through an educational domain, with the goal of combining the curricular structure of an engineered environment and the flexibility of a learner-centered setting. This paper describes an experiment in which our evolutionary method is used to supply content for keyboarding (typing) games in a web-based learning environment. Results are presented, based on data collected in a pilot study involving 44 children in a public primary school.


Netscape: CEL Keyit game
File Edit View Go Communicator Help
Bookmarks Location: http://www2.demo.cs.brandeis.edu/cel/
What's Related
CANCEL
NEW
EXIT
stop
skip

this is the game of KEYIT

...and this is your match with !

click CANCEL to cancel the match and return to the playground

click NEW to leave Keyit and enter a new playground

click EXIT to exit CEL

here comes the next word..

	owner	my score	your score
agglutinating		279.0	270.0
remoteness		289.0	257.0
ironing		151.0	145.0
conciseness			

type the word here ->

Figure 1: The game of Keyit.

2. The domain

In earlier work (Sklar and Pollack, 1998, Sklar, 2000), we built an Internet learning community for children called the Community of Evolving Learners (`http://www.demo.cs.brandeis.edu/cel`). CEL is located on a free web site and is open to anyone via a Java-enabled browser. Inside CEL, participants engage in multi-player educational games. The goal is for the system to adapt as the players learn and to provide continuous challenges for all players.

The work described here is based on a prototype version of CEL which contained two simple keyboarding (typing) games called Keyit and Pickey. These are both two-player games in which participants are each given 10 words to type and are scored based on speed and accuracy. For each player, a timer begins when she types the first letter of a word and time is marked when she presses the *Enter* key to terminate the word. Time is measured using the system clock on her computer, and her score is reported in hundredths of a second.

The two games are very similar to each other, differing mainly in user interface. One procedure is used to supply words to both games, so for simplicity, only Keyit is discussed here. Keyit is pictured in figure 1.

The words presented to Keyit players are selected from a database containing approximately 35,000 words. Every word in the database is characterized by a vector of seven feature values: (1) word length, (2) keyboarding level[3], (3) Scrabble^{TM} score[4], (4) number of vowels, (5) number of consonants, (6) number of 2-consonant clusters and (7) number of 3-consonant clusters. Each word can be thought of as a point in this 7-dimensional feature space. Words with similar feature values are considered to be close to each other in this space; words with disparate feature values are considered far away. Figure 2 illustrates this for the word `BLUE`, which has close neighbors `MEAT` and `BOIL`. The words `HIDE`, `DARK` and `RED` are further away, respectively.

Note that difficulty is not explicitly defined here, so that while we designate `RED` and `BLUE` as being far away from each other, we do not need to decide which is harder than the other. This lessens the amount of domain engineering then is required in an application that pre-defines curricular paths.

3. Word selection

An evolutionary approach is used to guide selection of words from the 7-dimensional feature space, geared to the changing needs of individual users. The basic evolutionary algorithm, from (Holland, 1975), is outlined in

[3]There are several standards which define an order for introducing keys to students learning typing. We used the ordering listed here: `http://www.absurd.org/jb/typodrome/`. We assign each word in the dictionary a number equal to the highest keyboarding level of any of the letters in that word.

[4]Scrabble^{TM} is a board game in which players take turns making interconnecting words by placing letter tiles on a grid in crossword puzzle fashion. Each letter is assigned a fixed value, calculated according to its frequency of usage in everyday American English. Players receive a score for each word they place, calculated by summing the values for each letter in the word.

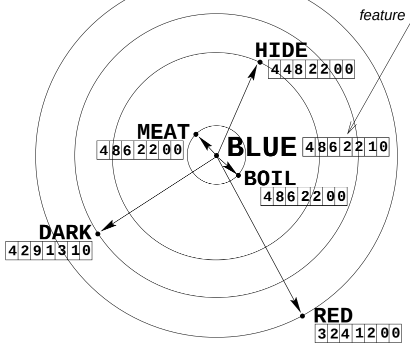


Figure 2: Distance between words in feature space.

table 1. The *elements* could be, for example, software agents exhibiting specific game strategies or various solutions to a hard search problem.

Table 1: Basic evolutionary algorithm.

1.	Initialize a *population* of randomly chosen *elements*.
2.	Let each element perform in the task domain.
3.	Evaluate each element's performance and, based on the evaluation, *select* some elements to be replaced.
4.	Produce a new population of elements, using *reproduction* techniques to replace the elements selected in step 3.
5.	Iterate, starting from step 2.

This algorithm is adapted to our task of selecting words for Keyit games. In this context, the *elements* of table 1 are words, and the population size is fixed at 10. The adapted method is shown in table 2. Note that this table contains a simplified form of the procedure, considering the needs of only one player; later in this section, the process is further modified to accommodate two players.

The selection and reproduction phases are illustrated in figure 3. The selection process (step 3 in table 2) involves comparing the score achieved for each word in G_t with the user's average score over all words encountered in games of Keyit. The idea is to partition G_t into two groups: those that the user knows how to type, and those with which the user needs more practice. "Score" is the total time it takes to type a given word. Words whose score is lower than the average are deemed "known" (faster is better); words whose score is higher than average are labeled "needs practice".

The reproduction phase (step 4 in table 2) entails replacing all the words in G_t with appropriate *children*, to get G_{t+1}. "Needs practice" words are replaced with others nearby in the 7-dimensional space, thereby exploiting regions with similar feature values to provide more opportunities to master the similar words while avoiding repetition, where the same words might be offered again and again until they have been learned satisfactorily. This is equivalent to making a small mutation to a word's feature vector. "Known" words are replaced by randomly jumping to some new area in the feature space, thereby exploring regions further away. This is equivalent to making a large mutation to a word's feature vector. The general idea is illustrated in figure 4.

The actual implementation of this procedure is complicated by two factors. First, when a game occurs between two human players, the set of ten words selected by the system must be appropriate for both players. Second, not all points in the 7-dimensional feature space are valid. If the reproduction phase used a standard operator like *mutation* or *crossover* (i.e. modifying one or more values in a parent's feature vector), the resulting vector would not necessarily correspond to a word in the dictionary. In fact, some combinations of feature values are invalid, e.g. word length must equal the number of

Table 2: Evolutionary word selection method.

1.	For a new user, initialize a population, $G_{t=0}$, of 10 randomly chosen words.
2.	For an old user, read the user's performance data P (which includes the population G_t of the 10 words from the user's last game).
3.	Evaluate the user's performance with the words in G_t, and, based on the evaluation, select entries that are "known" and entries that "need practice."
4.	Produce a new population of words, replacing all entries in G_t to get G_{t+1}. "Known" entries are replaced with words far away in the domain space, and "need practice" entries are replaced with words nearby.
5.	Supply G_{t+1} to the user's applet for the current game.
6.	Iterate, starting from step 2, when the next game occurs.

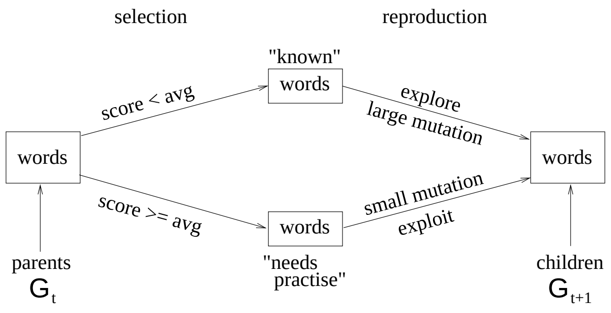


Figure 3: Selection and reproduction.

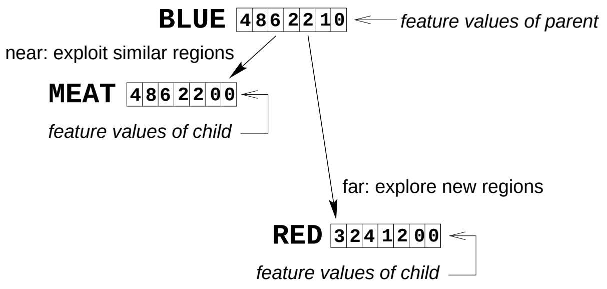


Figure 4: Exploitation and exploration in feature space.

vowels plus the number of consonants. To address these complications, two procedures are introduced: *merging* and *reproduction through sampling*.

3.1 Merging

The merging procedure is implemented so that the contents of G_{t+1} is appropriate for both players engaged in the match. The basic process involves combining the user performance data for both players and creating a third, composite player that is essentially an average of the two humans' performance statistics. For each user, a performance record P is stored, which includes scores for all words previously encountered, the population G_t of the 10 words from the user's last game and an average score (avg) over all words the user has seen.

In table 2, steps 1 and 2 are modified to read performance data for both players and then to merge the data, so that steps 3 and 4 are performed using this composite data set. The merge process essentially consists of concatenating the performance data for both players and computing a composite average, which is simply the average of the two avg values from each player's P. Finally, step 5 sends the new population of words to both users' applets. At the end of the merge process, it is guaranteed that the composite G_t contains at least 10 and at most 20 words.[5] At the end of the reproduction process, G_{t+1} will contain exactly 10 words.

3.2 Reproduction through sampling

The reproduction procedure must be able to take a parent and produce a child whose feature values are either near to or far from those of its parent, corresponding to *exploitation* (small mutation) and *exploration* (large mutation), respectively (as illustrated in figure 3). In theory, a traditional reproduction method like *mutation*[6] could be used for both tasks. To find a nearby entry in the feature space, one of the parent's feature values could be selected at random, and incremented or decremented, to result in a new vector with only one value different from the parent vector. To find an entry far away in feature space, more of the parent's feature values could be altered, resulting in a new vector with values distinct from its parent.

As mentioned earlier, the problem with using this technique in this domain is that the new vector would not necessarily be valid or correspond to a word in the dictionary. Some applications of evolutionary algorithms handle this kind of situation by applying a correction to the reproduction operator, ensuring that the result is valid. For example, a mathematical function (i.e. *modulo*) might be used to force the mutation of an individual feature value to fall within a specified numeric range. The situation here is complicated by the fact that even if individual feature values are valid, when taken in combination, the entire vector may be invalid. A simple method for overcoming this problem would be to try a series of mutations iteratively, stopping when a valid vector was found.

However, with this particular domain, the 7-dimensional feature space is quite sparse. If bounds are considered on each feature value (for example, word length must be between 2 and 25 characters, and keyboarding level must be between 0 and 10), then there are over 90 million possible combinations of feature values. Yet the dictionary used here only accounts for 6074 of those combinations, less than 0.0065%. This means that the likelihood of a mutation producing an invalid set of feature values is prohibitively high. An iterative procedure like the simplistic one mentioned above could take a long time to run. Because our evolutionary method operates in a real-time environment, where the customers are (impatient) children, minimizing run-time is vital. A target maximum of 1 second was chosen for the procedure to run in its entirety.

One approach to the sparse feature space problem would be conceptually to mutate from one entry in the domain to another, rather than from one vector to another. As indicated by figure 2, all words in the domain can be represented as points in 7-dimensional space, thus it is possible to sort the entire dictionary according to the entries' feature values. This would mean computing a 35000×35000[7] matrix containing the distance in feature space from each entry to every other entry. Then when making mutations, the procedure need only look up entries in this matrix — small mutations would look for close neighbors and large mutations would look further away. However, again, practical considerations ren-

[5]There may be less than 20 words in G_t because duplicates are eliminated when this set is created, so if there were any commonalities between the words of the individual players' last games, then these would be removed and $|\mathsf{G}_t|$ would be less than 20.

[6]For simplicity, the discussion here is limited to mutation. Crossover or other gene altering methods could also be used, but the problems encountered with using mutation in the present domain and real-time environment (as detailed in this section) would also occur with these other methods.

[7]The size of the database is approximately 35000 entries. Since the distance between any two entries is symmetrical, the size of the matrix really need only be $(35000 \times 35000)/2$.

der this solution infeasible because too much memory is required to store this matrix[8].

An alternative to storing the entire table in memory would be to load the relevant portion from disk during run-time; however testing proved that selective loads took longer than the 1 second time requirement. Another option would be to compute the relevant portion of the table during run-time; again, testing showed that this method exceeded the maximum time requirement.

The solution we settled on was to adopt a new reproduction process called *reproduction through sampling.* The strategy is to begin by randomly selecting a relatively small sample population from the dictionary and then to replace the parents in G_t with children chosen from this sample. The process is defined as follows and is illustrated in figure 5.

1. Select a random sample of 1000 words from the 35,000 word dictionary:
```
for i ← 1 to 1000
  x ← random( 1,35000 ) - 1;
  $S_i$ ← $dictionary_x$;
```

2. Compute the distance between entries in S and G_t:
```
for j ← 1 to 1000
  for i ← 1 to $|G_t|$
    $dist_{i,j} \leftarrow \lceil (\sum_{k=1}^{7} [\frac{(g_{i,k} - s_{j,k})}{(max_k - min_k)}]^2) * c \rceil$;
```

3. Sort each row in the distance matrix:
```
for i ← 1 to $|G_t|$
  sort( $dist_i$ );
```

4. Compute the mean for each row in the distance matrix:
```
for i ← 1 to $|G_t|$
  $meandist_i$ ← mean( $dist_i$ );
```

5. For each row in the distance matrix, save the index of the value closest to the mean:
```
for i ← 1 to $|G_t|$
  $xmeandist_i$ ←index of entry in $dist_i$ whose
              value is closest to $meandist_i$;
```

6. Generate a child for each element in G_t:
```
for i ← 1 to $|G_t|$
  if $g_i$ has a score
    if ( $g_i$.score < avg ) then
      j ← pick unused element in $S_i$ from
          FAR end of dist (explore);
    else
      j ← pick unused element in $S_i$ from
          NEAR end of dist (exploit);
  else
    j ← pick a new word randomly from $S_i$;
  $g_i \leftarrow s_j$;
```

[8] "Too much" simply means more than is available on the CEL server for use by this process.

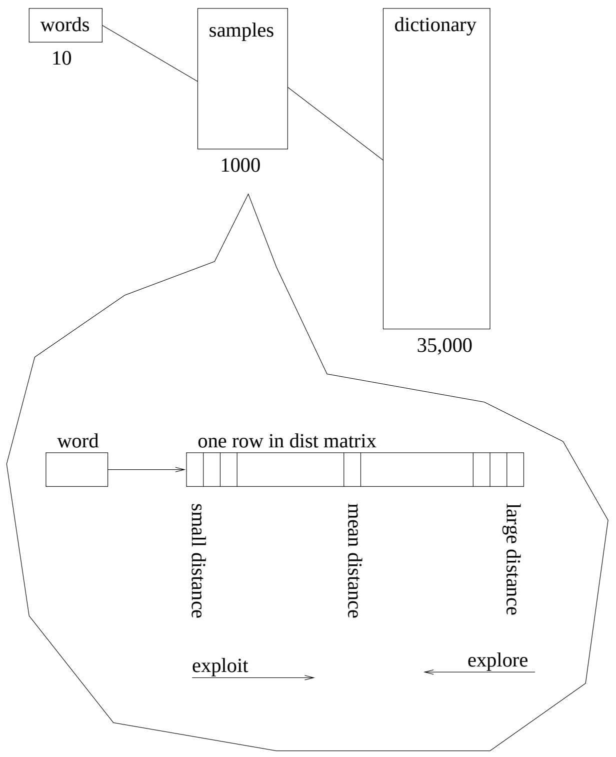


Figure 5: The reproduction algorithm.

Table 3 illustrates the relationship between one parent and one child word list. An integer distance was used (during the pilot study) because of memory limitations on our initial server. However, a more capable server is in use now and future work will compare use of a real-valued mean-squared distance.

4. Results

In early 1999, we conducted a pilot study where 44 fourth and fifth grade students used the two typing games in CEL (Keyit and Pickey) over a period of five months. Data collected in this study is presented here. The domain coverage for each user was examined, to determine if the evolutionary word selection method led players into more of the domain space than a pre-leveled application might. Additionally, the relationship between typing speed and various word features was analyzed, to determine which features, if any, emerged as more highly correlated (to typing speed) than others. Finally, the change in typing speed for each participant, as tracked by our system, is shown.

4.1 Domain coverage

Of the seven feature values that define the domain, only Scrabble score and keyboarding level are examined here, since the remainder are also a function of word length (as is Scrabble score) and so can be considered redundant in

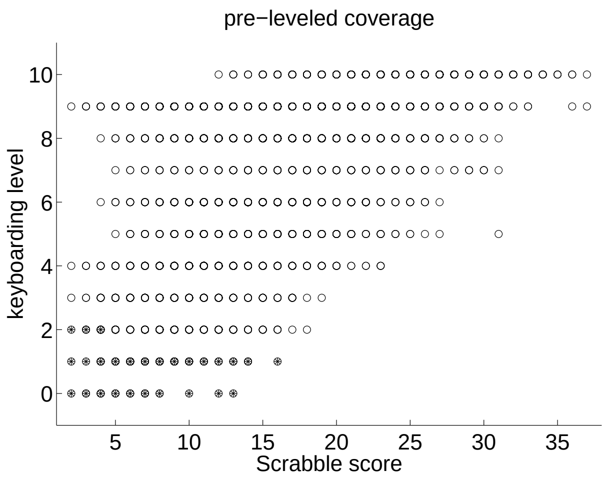
Table 3: Distance between corresponding words from one parent and child generation.

	parent	child	dist
1	four	four	
	[4,3,7,2,2,0,0]	[4,3,7,2,2,0,0]	0
2	who	aim	
	[3,5,9,1,2,1,0]	[3,8,5,2,1,0,0]	1
3	race	peas	
	[4,7,6,2,2,0,0]	[4,6,6,2,2,0,0]	1
4	vies	dates	
	[4,7,7,2,2,0,0]	[5,4,6,2,3,0,0]	2
5	away	fives	
	[4,5,10,2,2,0,0]	[5,7,11,2,3,0,0]	2
6	singed	calorie	
	[6,9,8,2,4,1,0]	[7,7,9,4,3,0,0]	2
7	forked	debated	
	[6,3,14,2,4,1,0]	[7,8,11,3,4,0,0]	3
8	enumerates	ragged	
	[10,9,12,5,5,0,0]	[6,3,9,2,4,1,0]	2
9	manipulated	perused	
	[11,9,16,5,6,0,0]	[7,6,10,3,4,0,0]	1
10	fosters	numerics	
	[7,4,10,2,5,2,0]	[8,9,12,3,5,1,0]	3

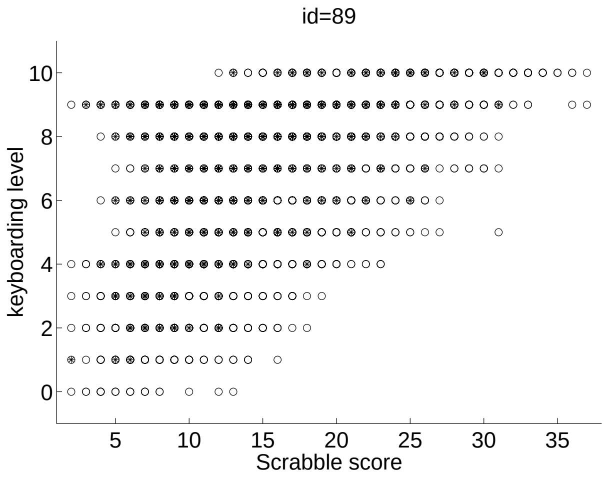
this analysis.

The chart in figure 6(a) is a sample domain coverage chart, plotting Scrabble score versus keyboarding level. A point exists in the domain space for each circle on the plot. For each point that a user has been exposed to, the circle is filled (•). Thus the open circles (○) represent portions of the domain space that the user has not seen. This sample chart illustrates the coverage that a user might experience in a pre-leveled environment, where (e.g.) she must complete all problems in keyboarding level 0 before seeing any in keyboarding level 1.

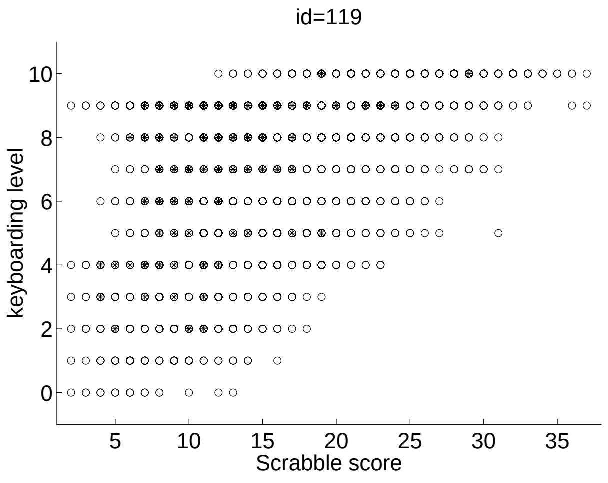
Figures 6(b) and 6(c) contain domain coverage charts for two of the students involved in the pilot study. Both students have been exposed to a large portion of the domain. Student 89, who is the fastest typer, has seen more of the domain than the slowest typer, student 119. This result illustrates that the system adapts according to the capabilities of the individual user, showing more of the domain to the student who is ready to see it. As well, the students' experiences are more varied than with a standard pre-leveled curriculum, as figure 6(a) exemplifies.



(a) sample



(b) id = 89 (fast)



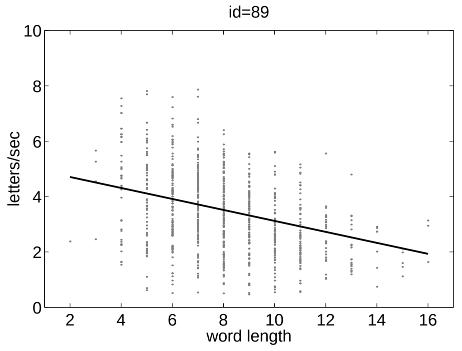
(c) id = 119 (slow)

Figure 6: Domain coverage charts.

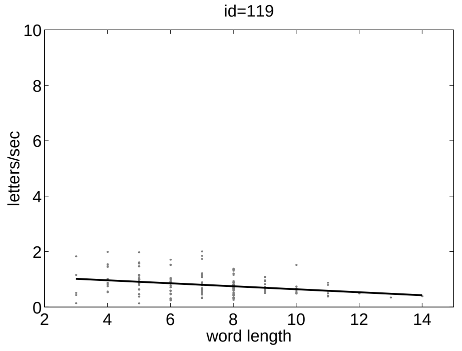
4.2 Feature correlation

The relationships between word length, Scrabble score, keyboarding level and typing speed are examined here, in order to ascertain if any one of these three features appears to correlate more closely with typing speed than any of the others. There should be a direct correlation between word length and typing speed. It has been demonstrated that an artifact of typing long words exists such that the speed per letter is slower than for shorter words (Larochelle, 1982). Thus, even when the time it takes to type a word is normalized for the length of the word, so that speed is measured in letters per second, longer words still take more time to type than shorter words.

Figure 7 shows plots for the same two students mentioned above. On each graph, there is a point for each word typed by the corresponding student. The straight line is a linear least-squares fit of all the points. The artifact (i.e. longer words take longer to type) is readily apparent in the figure.



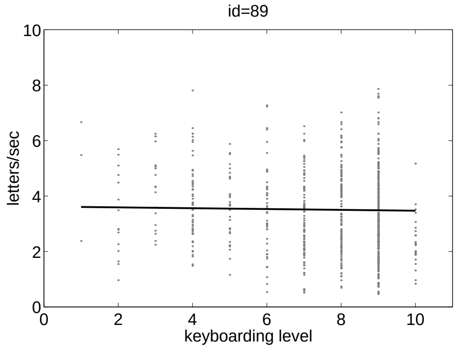
(a) id = 89 (fast)



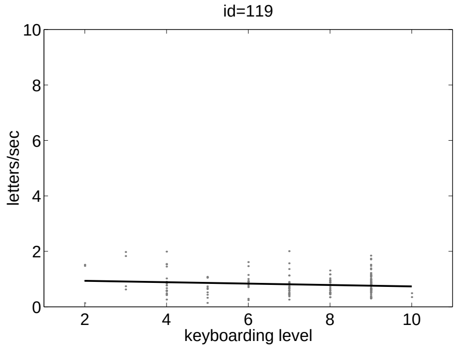
(b) id = 119 (slow)

Figure 7: Word length vs typing speed.

Figures 8 and 9 show the relationships between keyboarding level and Scrabble score with typing speed, for the same pair of students.

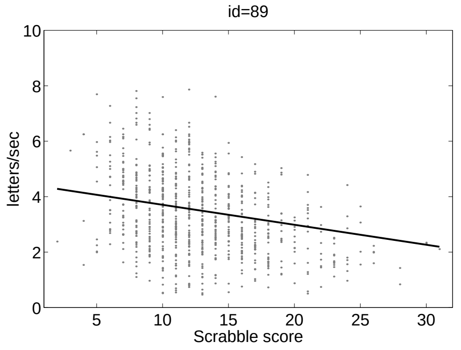


(a) id = 89 (fast)

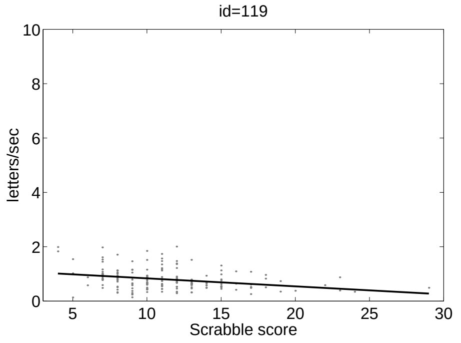


(b) id = 119 (slow)

Figure 8: Keyboarding level vs typing speed.



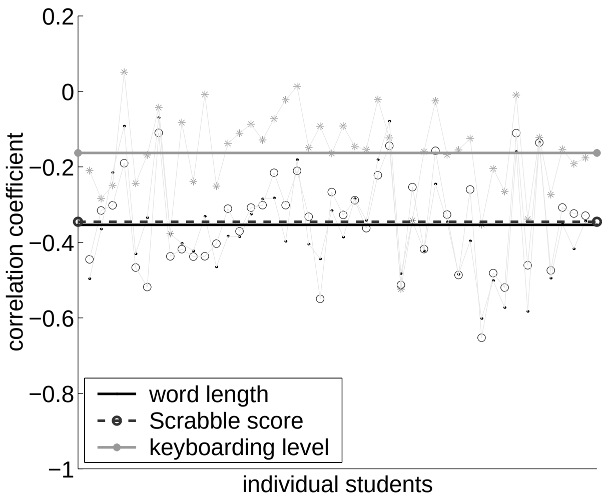
(a) id = 89 (fast)



(b) id = 119 (slow)

Figure 9: Scrabble score vs typing speed.

Figures 7 through 9 appear to indicate that typing speed correlates more directly with word length and Scrabble score than with keyboarding level. Figure 10 shows the correlation coefficient, for each user, between typing speed and each of these features. Each point on the chart corresponds to one of these statistics per user (i.e. there are three points per user). Faint lines connect the points for each feature, to make it easier for the viewer to group the points. Horizontal lines are drawn to indicate the mean correlation coefficient for each feature, across all users. A correlation coefficient closer to -1 indicates a higher negative correlation between two variables — e.g. that longer word length is indicative of slower typing speed, per letter.



	mean	stdev	median
word length	-0.353	0.130	-0.371
Scrabble score	-0.345	0.128	-0.327
keyboarding level	-0.163	0.116	-0.154

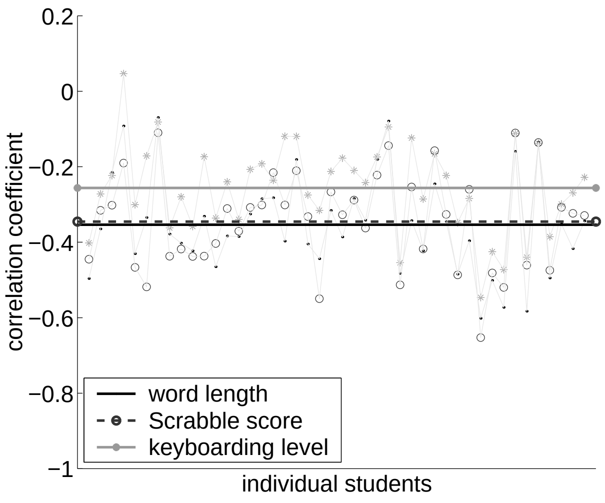
Figure 10: Correlation coefficients, for all users.

Scrabble score is a function of word length, since each letter in the word contributes to the score individually. Conversely, keyboarding level is computed independently of word length. So it is not surprising that word length and Scrabble score exhibit similar statistical characteristics. Indeed, the higher correlation for word length dependent statistics and typing speed confirm the statement made in the previous section: that long words take more time to type, on a per letter basis, than short words.

For comparison, a modified keyboarding level was computed in which keyboarding level is also defined as a function of the length of the word (as is Scrabble score and obviously word length). Instead of calculating keyboarding level to be chosen as the highest level of any letter in a word, the level of each letter in the word was totalled (the same way that Scrabble score is tallied).

This accumulative keyboarding level was computed for all the data, after the case study was finished. The purpose was to determine if keyboarding level really correlated so poorly to typing speed as was indicated in figure 10, or if the method of computing keyboarding level (where word length was not a factor) skewed the correlation results away from keyboarding level. After all, since the data was collected during keyboarding games, it would seem logical that keyboarding level should be highly correlated to typing speed.

Figure 11 shows the correlation results with the accumulative keyboarding level. The mean correlation coefficient for accumulative keyboarding level is -0.256, better than that with the original keyboarding level (-0.163). However, word length and Scrabble score still correlate significantly higher than keyboarding level. This is a very interesting result. The conclusion drawn is that in future, words should be chosen more on the basis of word length and Scrabble score than keyboarding level.



	mean	stdev	median
word length	-0.353	0.130	-0.371
Scrabble score	-0.345	0.128	-0.327
keyboarding level	-0.256	0.119	-0.241

Figure 11: Correlation coefficients, for all users, with accumulative keyboarding level.

4.3 Learning

Finally, we look at the change in the children's typing speed between the beginning and the end of the study. The children engage in many keyboarding activities, so we cannot attribute their progress exclusively to their usage of CEL. However, we can track their progress, as illustrated in figure 12. Most of the children (85%) improved, shown as a (positive) increase in typing speed.

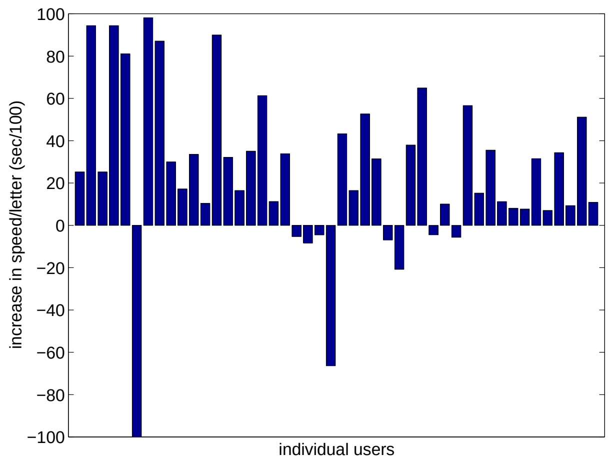


Figure 12: Change in typing speed.

5. Discussion

The primary advantages of using an evolutionary approach to guide problem selection in an educational game include:

- Students guide themselves through the domain, based on their own performance with the system, which means that the system adapts in real-time to the needs of each individual.

- Students may reach areas of the domain that they might not see otherwise, where a standard pre-leveled system may prevent them from leaving an area before it is completely mastered.

- Costs and effort to implement the game are reduced, because the domain paths need not be pre-defined in the way that structured educational systems require.

The results presented here indicate that the use of an evolutionary approach is a viable alternative to pre-leveled methods. In future work, control studies must be performed in order to validate this statement. For example, some students' games would be supplied with words chosen by the evolutionary method, others would be chosen at random and others would be chosen according to a standard pre-leveled curriculum.

It has been suggested that frequency of word usage in the English language should also be a feature in the domain space for word games. Although this is partially encoded in the frequency of letter count that is part of Scrabble score, use of a precise statistic may be beneficial. While touch typing courses commonly require learners to type non-linguistic sequences of characters, people will generally say that they can type words they know faster than words for which they do not know the meaning. Future work will involve adding this dimension to the feature space and studying the resulting correlations.

The next step is to apply this technique to more sophisticated applications and more complex domains. We are currently building broader activities, including a spelling bee and a collaborative anagrams game, using the same method for supplying game content. Additionally, we are investigating using our method with domains that do not lend themselves to leveling, such as geography. Here, engineering paths through a curriculum is difficult and so our method may prove particularly useful.

Finally, it may be considered that the evolutionary approach described here is actually co-evolutionary. The evaluation stage, wherein words are designated to be replaced by exploration or exploitation, does not use a fixed function, as does a standard evolutionary algorithm. The evaluation is made by comparing the typing speed for each word in the parent generation to the user's overall average typing speed. As the user advances, her typing speed increases; and this increase happens at different rates for different users. Thus the evaluation is a relative one — the word lists evolve along with the users who type them.

6. Acknowledgements

Thanks to Alan Blair for help with things mathematical, to Richard Watson for advice on the evolutionary content and to Simon Levy for constructive comments on the text. This work was supported by the Office of Naval Research under N00014-98-1-0435.

References

Beck, J. (1997). Modeling the student with reinforcement learning. In *Proceedings of the Machine Learning for User Modeling Workshop, Sixth International Conference on User Modeling.*

Bruckman, A. (1997). *MOOSE Crossing: Construction, Community, and Learning in a Networked Virtual Community for Kids.* PhD thesis, MIT.

Fanderclai, T. (1995). Muds in education: New environments, new pedagogies. *Computer-Mediated Communication Magazine*, 2(1).

Forrester, J. (1992). System dynamics and learner-centered-learning in kindergarten through 12th grade education. Technical Report D-4337, MIT.

Gordon, A. and Hall, L. (1998). Collaboration with agents in a virtual world. In *Workshop on Current Trends and Applications of Artificial Intelligence in Education: 4th World Congress on Expert Systems.*

Greer, J. E. and McCalla, G. I., (Eds.) (1994). *Student Models: The Key to Individualized Educational Systems.* Springer Verlag, New York.

Holland, J. H. (1975). *Adaption in Natural and Artificial Systems.* University of Michigan Press.

Larochelle, S. (1982). A comparison of skilled and novice performance in discontinuous typing. In Cooper, W., (Ed.), *Cognitive Aspects of Skilled Typewriting*, pages 67–94, New York. Springer-Verlag.

Papert, S. (1993). *The Children's Machine.* BasicBooks.

Resnick, M. (1997). *Turtles, termites, and traffic jams: explorations in massively parallel microworlds.* MIT Press.

Shiri-A., M., Aïmeur, E., and Frasson, C. (1998). Case-based student modelling: unaccessible solution mode. In *Conference internationale sur les nouvelles technologies de la communication et de la formation (NTICF'98).*

Sklar, E. (2000). *CEL: A Framework for Enabling Experimentation in an Internet Learning Community.* PhD thesis, Brandeis University.

Sklar, E. and Pollack, J. B. (1998). Toward a community of evolving learners. In *Proceedings of the Third International Conference on the Learning Sciences (ICLS-98).*

VanLehn, K., Niu, Z., Slier, S., and Gertner, A. (1998). Student modeling from conventional test data: A bayesian approach without priors. In *Proceedings of the 4th Intelligent Tutoring Systems Conference (ITS'98)*, pages 434–443.

Walters, J. and Hughes, B. (1994). Camp marimuse: Linking elementary and college students in virtual space. In *Proceedings of the National Educational Computing Conference.*