A System for Generating, Archiving, and Retrieving Specialized Assignments Using LaTeX

Stina Bridgeman
Brown University
ssb@cs.brown.edu

Michael T. Goodrich
Johns Hopkins University
goodrich@jhu.edu

Stephen G. Kobourov
Johns Hopkins University
kobourov@cs.jhu.edu

Roberto Tamassia
Brown University
rt@cs.brown.edu

Abstract

We present SAIL, a package for the creation of Specialized Assignment In LaTeX. We describe several features which allow an instructor to create sufficiently different instances of the "same" problem so as to encourage student cooperation without fear of plagiarism. The SAIL package also provides support for grading aids and grading automation. In addition, we describe an on-line system for archiving homework problems in a database that can be easily searched and to which new parametrized problems can be easily added. Together, the SAIL package and the searchable database of problems offer a powerful tool for generating, archiving, and retrieving homework assignments (as well as tests and quizzes).

1 Introduction

Recent education research has shown the value of assigning homework problems that are specialized so that each student is given a different problem that tests knowledge of the same subject matter. Such specialized problems enable an instructor to allow students to collaborate without fear of plagiarism, for discussions between students must necessarily be at a higher level than simply answer syntax. That is, it is not operative for a student to ask another, "What was your answer to Problem number 7?" A more appropriate question would be, "Could you remind me how QuickSort works?" or "Would you please teach me how deletion in a red-black tree works?"

Given the proven value of specialized homework assignments, we feel that the next natural issue to address is the development of tools to allow instructors to easily prepare specialized assignments. This paper describes one such tool, which we call the SAIL package. In addition to this tool for generating specialized problems, we have also developed an on-line system for archiving and retrieving such problems. Thus, we describe a complete system that allows an instructor (or group of instructors and teaching assistants) to easily assemble complete sets of assignments, exams, and quizzes that include specialized problems.

1.1 Previous Work

As mentioned above, recent research has shown the benefit of individualized assignments as a means of achieving higher level of collaboration among students. Examples include research on their use in chemistry classes [10], statistics classes [11], and introductory computer science courses [12]. This prior work points to evidence that such use of specialized assignements provides better learning of fundamental material rather than aiming to get simply a correct answer.

Our approach of eliminating the usefulness of plagiarism contrasts with some interesting prior research on creating tools that can detect plagiarism. This detection approach naturally assumes that assignments are identical and attempts to detect plagiarism rather than prevent it. Examples of such programs include the Unix commands `diff` [5], which uses string alignment methods to detect similarity between arbitrary text files and `dup` [2], which finds exact matches over a given length. In addition, several tools have been designed with the specific goal of detecting plagiarism in computer science assignments, such as `moss` [1], `sim` [3], and the programs of [6, 7].

1.2 Our Results

In this paper we present a simple package of LaTeX macros, which we call SAIL, that provides tools for instructors to generate differing versions of computer science problems. This system allows problems given to all the students in a class to be of the same type but differ sufficiently in details. Thus in order to "help" a friend in class, a student cannot simply hand him her own assignment, but she would rather have to explain her solution to him.

We chose to develop our assignment generating system using the LaTeX system for a number of reasons. First, many computer science faculty already use LaTeX to create homework assignments, quizzes, and exams (as well as SIGCSE submissions such as this one). Thus, building on the LaTeX system is natural for many instructors in computer science. Second, the LaTeX document preparation system is *Turing equivalent*; that is, it is possible to write a LaTeX source file to compute the value of any computable function. This power is more than of theoretical interest, however, for it provided us with the ability to write the `SAIL` package to include macros for pseudo-random number generators, pseudo-random permutations, the ability to select and read files at random, etc. Finally, we chose the LaTeX system because it is designed to be platform independent. It is still one of the few document preparation systems that supports complete source-file interoperability between Mac, Unix/Linux, and Windows computer systems, and there are many freeware versions of LaTeX available for all of these systems. Moreover, because it is platform independent, LaTeX has few possibilities for interacting with operating systems in dangerous ways (e.g., as the Melissa virus did inside Microsoft Word documents).

In addition, since LaTeX source files are written in ASCII, our choice of using LaTeX greatly simplified the second component of our system, which is an on-line problem set repository. Our on-line system is capable of archiving and retrieving homework assignments containing specialization macros and/or standard LaTeX commands. Our on-line system also allows for keyword searching of multiple files and directories, and it returns its results in the form of both LaTeX source and example output in PDF. Thus, we present a complete solution for generating, archiving, and retrieving specialized assignments.

2 The `SAIL` Package

To provide the capability of generating different versions of similar problems we need the ability to generate (pseudo-)random numbers in LaTeX. We achieved this ability by building on the existing `fp` package by Michael Mehlich for floating point operations, which includes a LaTeX macro for generating a (pseudo)-random floating point number, and the `exam` package by Hans van der Meer, which includes a LaTeX macro for generating a random bit. Neither of these systems were completely sufficient for our uses, however, as the `fp` system does not include integer operations or list manipulation methods, and the `exam` package only includes random bit generation and random list permutations for multiple-choice questions with at most five (5) possible answers. Rather than describe our `SAIL` package in detail, however, we provide some examples of its use in this and the following sections, including how to incorporate it with our on-line problem set repository.

2.1 Random Sequence

In the `SAIL` package we can use the macro `\rsequence#1#2#3` to generate a sequence of $n =$ #3 random numbers in the range [#1, #2). This macro can be very useful in problems such as sorting. In particular, consider the following problem in LaTeX:

```
\begin{question}
Sort the following numbers using
MergeSort and use a binary tree to show
the input and output of each recursive
call:
\rsequence{0}{100}{10}
\end{question}
```

The problem above will produce different sequences of 10 random numbers in the range [0, 100). Here is one output:

> **Question 1** *Sort the following numbers using MergeSort and use a binary tree to show the input and output of each recursive call:*
> *94 7 10 40 84 13 18 32 85 32*

2.2 Sequence Without Repeats

Often, it is necessary that the input sequence to a given problem contain no repeats. This sometimes simplifies a problem as in the case of insertion into a binary search tree. With this in mind we created the `\rsequencenr#1#2#3` macro. This macro is very similar to the `\rsequence` macro defined above; the only difference is that there will be no repeated numbers in the produced sequence. Consider the following problem in LaTeX:

```
\begin{question}
Show the result of inserting the
following numbers in a binary search
tree:
\rsequencenr{10}{50}{15}
\end{question}
```

The problem above will produce different sequences of 15 unique random numbers in the range [10, 50). Here is one possible output:

> **Question 2** *Show the result of inserting the following numbers in a binary search tree:*
> *17 39 15 22 10 48 33 29 11 32 43 37 30 19 44*

2.3 Random Permutation

The `exam` package provides the choice of permuting up to five answers to a multiple choice question. We created a macro `\perm#1#2` which produces a random permutation of the numbers [#1, #2). This macro can be used in shuffling multiple choice answers. There is no limit to the number of answers that can be permuted. It can also be used in sorting problems and in other problems in which repeats are undesirable.

Here is an example of a multiple choice question:

```
\begin{question}
What does ``AVL'' stand for in AVL-Trees?
\end{question}
\begin{choice}
\item American Veteran Labeling
\item Aggregate Virtual Leaf
\item First initials of its two creators
\item AVerage Left-balance
\end{choice}
```

Here is a possible outcome:

> **Question 3** *What does "AVL" stand for in AVL-Trees?*
>
> a) Aggregate Virtual Leaf
>
> b) First initials of its two creators
>
> c) AVerage Left-balance
>
> d) American Veteran Labeling

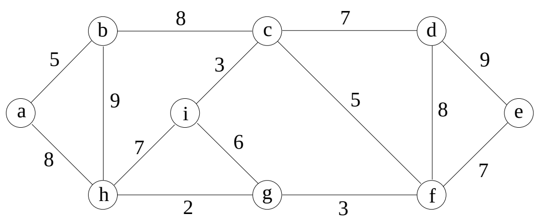
2.4 Choosing a Random File

A large class of problems can be parametrized by the inclusion of a random graph. Consider the Minimum Spanning Tree problem and assume we have generated N random graphs on the same number of vertices. Let the files be stored as $graph0, graph1, \ldots, graphN$ and let files $algorithm0, algorithm1, algorithm2$ contain the words "Boruvka's", "Prim's", "Kruskal's", respectively. Consider the following question:

```
\begin{question}
Highlight the edges of an MST of the
following graph using
\newcount\temp \rnumber{3}{\temp}
\input\namenumber{algorithm}{\temp}
algorithm:
\rnumber{\N}{\temp}
\input\namenumber{graph}{\temp}
\end{question}
```

The problem above will select a random graph and assign one of the three algorithms for finding an MST. Here `\rnumber#1#2` assigns a random number in the range [0,#1) to #2 and the macro `\namenumber` attaches number #2 to the file name #1. (the definition is `\def\number#1#2{#1\the#2}`). Here is a possible outcome:

> **Question 4** *Highlight the edges of an MST of the following graph using Prim's algorithm:*



2.5 Other Problems

The random number generator can be used for other user-defined macros or by itself in a variety of problems. Consider the Shortest Path problem (and all of its derivative problems). The input to the problem is generally a weighted undirected graph. The instructor can create one unweighted graph with node labels. Then a list of weights can be generated at random with the random number generator. As can easily be seen, this approach is extendable to many graph and tree based problems (e.g. tree traversal Algorithms) and many others.

3 Using the SAIL Package

For a LaTeX package to be most effective and useful it has to be simple to use, flexible, and easy to build upon. We have created only a handful of macros which are easy to use and work with. There are also several options which provide for:

- reproducible random assignments
- support for graders
- automated grading

3.1 Customizing the SAIL Package

By setting the macro, `\csize` to the class size, the instructor can generate `\csize` different versions of the given assignment at once. With this in mind the LaTeX source of a homework assignment is of the following form:

```
\documentclass{article}
\usepackage{SAIL}
...
\begin{document}
\loop\ifnum0<\csize{
  ...
  body of the assignment
  ...
  \setcounter{page}{1}
  \newpage
  }
  \repeat
\end{document}
```

While this approach provides sufficient randomness, the assignments cannot be repeatedly reproduced. Therefore, the instructor can use the student id's as keys for the random number generator to generate assignments which can be regenerated again if necessary. There are two ways in which this can be achieved. A file containing all of the student id's can be created and used in the seed initialization, or LaTeX can prompt the user for the student id number.

3.2 Grading with SAIL

One of the major drawbacks of using different assignments is the increased difficulty in grading. With this in mind, we have provided the capability to create an auxiliary document which contains the random instances, along with an id number for the assignment (student id's if they were used in the initialization, or randomly assigned id's otherwise). The auxiliary file can be used as grader aid as follows. For a given assignment there could be a program which takes as input the random instances of the problems in the assignment and uses them to create solutions. Each auxiliary file can then be used to create solutions. Grading the student's assignments with valid solutions in hand can greatly aid the teaching assistant and in some cases even allow for grading automation. For example, in the case of multiple choice questions, the auxiliary file can even contain the answer key.

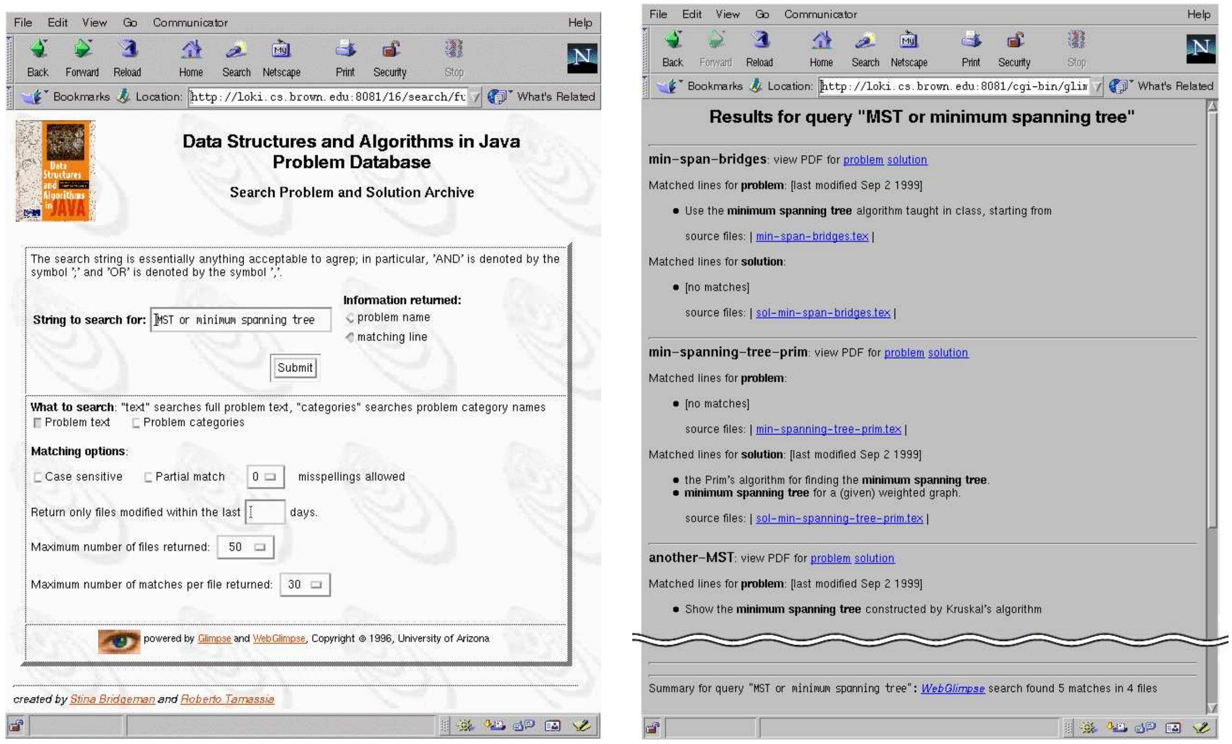
4 Archiving and Retrieving Problems

Homework creation can be further facilitated by having a searchable database of problems from which to draw. A prototype database[1] has been developed to accompany the book *Data Structure and Algorithms in Java* [4]. The database uses the GLIMPSE system [9] to allow fast full-text searching. Problems may also be classified into one or more categories, and retrieved by category. The database is accessed via the WWW; Figure 1(a) shows the query form. A great deal of flexibility is allowed in the search string in terms of wildcards; GLIMPSE's syntax is similar to that of `agrep` [8].

Figure 1(b) shows the result of searching the database for problems containing the text "MST" or "minimum spanning tree". For each problem, the lines matching the query string are displayed to give the user some context from which to narrow down the search for an appropriate problem. Links are given to PDF versions of the problem and solution, and to the LaTeX source and any other necessary accompanying files (such as figures). Providing the source files makes it possible to build homework assignments by cutting-and-pasting problems from the database. The solution files are password-protected, so while the links will always be shown, access to the solutions themselves is limited to instructors and other qualified individuals.

Problems using the SAIL package can be incorporated into the database like any other problem — the contents of all of the problem's LaTeX source files contribute to the index for that problem. As a

[1] available at `http://loki.cs.brown.edu:8081/16/`



(a) submisison form (b) result page

Figure 1: The problem database.

result, the Minimum Spanning Tree example in section 2.4 will be returned as a match for the search strings "Boruvka", "Prim", or "Kruskal" as well as "MST". The only part of the problem that will not be indexed are the randomly generated numbers, but numbers are not indexed in the first place.

References

[1] A. Aiken, Measure of Software Similarity, *htpp: www.cs.berkeley.edu/˜aiken/moss.html.*

[2] B. S. Baker, Parametrized Pattern Matching: Algorithms and Applications, *J. Comput. System Sci.*, 52 (1996), 28–42.

[3] David Gitchell and Nicholas Tran, Sim: A Utility for Detecting Similarity in Computer Programs, *Proceedings of the 30th SIGCSE Technical Symposium*, (1999), 266–270.

[4] Michael T. Goodrich and Roberto Tamassia, *Data Structures and Algorithms in Java*, John Wiley & Sons, New York 1998.

[5] J. W. Hunt and M. D. McIllroy, An Algorithm for Differential File Comparison, *Technical Report 41, Bell Labs*, (June 1976).

[6] H. T. Jankowitz. Detecting Plagiarism in Student Pascal Programs, *Computer Journal*, 31 (1988), 1–8.

[7] L. Malmi, M. Henrichson, T. Karras, J. Saarhelo, and S. Saerkilahti, Detecting Plagiarism in Pascal and C Programs, *Technical Report, Helsinki University of Technology*, (1992).

[8] Sun Wu and Udi Manber, agrep - A Fast Approximate Pattern-Matching Tool, *Proceedings of the Usenix Winter 1992 Technical Conference*, 1991, 152–162.

[9] Sun Wu and Udi Manber, GLIMPSE: A Tool to Search Through Entire File Systems, *Proceedings of the Usenix Winter 1994 Technical Conference*, 1994, 23–32.

[10] Morrisey, Kashy, and Tsai, Personalized Assignments for Freshman Chemistry, *Journal of Chemical Education*, 72 (1995), 141–146.

[11] Richard L. Rogers, A Microcomputer-based Statistics Course with Individualized Assignments *Teaching Psychology*, 12 (1987), 109–111.

[12] B. Toothman and R. Shackelford, The Effects of Partially-Individualized Assignments on Subsequent Student Performance, *Proceedings of the 29th SIGCSE Technical Symposium*, (1998), 287–291.