Use of an Algorithmic Search Tool in an Educational Setting

Christian Collberg, Stephen G. Kobourov, Suzanne Westbrook
Department Computer Science
University of Arizona
{collberg,kobourov,sw}@cs.arizona.edu

Categories and Subject Descriptors

D.2.2 [**Software Engineering**]: Design Tools and Techniques—*Software libraries*; D.2.4 [**Software Engineering**]: Software/Program Verification—*Assertion checkers*; H.5.2 [**Information interfaces and presentation**]: User Interfaces—*Graphical user interfaces*

General Terms: Algorithms, Languages, Verification

Abstract

AλgoVista is a web-based search engine that assists programmers to find algorithms and implementations that solve specific problems. The search engine is not keyword based but rather requires users to provide (input $\Longrightarrow$ output) samples that describe the behavior of their needed algorithm.

AλgoVista is easy to use. To search for a particular algorithm or classify a combinatorial structure a user simply *draws* the query in a drawing pane on a web browser. The result of the search is a list of links to web resources describing or providing implementations of the algorithm.

AλgoVista has many interesting applications in an educational setting. The search engine can help research students classify obscure problems and locate algorithms that would otherwise be hard to find in textbooks. Students can also add calls in their own programs to AλgoVista's database of executable problem specifications in order to dynamically check the correctness of their programs. Finally, instructors can use AλgoVista to set novel assignments in algorithms and data-structure classes.

This paper reports on the use of AλgoVista in two algorithms and theory classes, one at the undergraduate and one at the graduate level.

1. Introduction

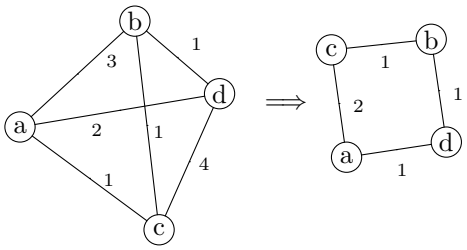
AλgoVista is a novel, web-based search engine aimed at helping programmers locate theoretical results from Computer Science. A programmer can simply *draw* an example of the behavior of an algorithm they are looking for and AλgoVista will search its database for problems that match this behavior. The database currently contains some 350 problem descriptions from the graph, numeric, and geometric domains.

The example in Figure 1 shows a situation where a programmer is looking for a problem that "computes the shortest path through all the cities on a map." The query is formulated by drawing a weighted graph and indicating that the function the programmer is looking for should map this graph to the number 5. AλgoVista replies that this looks like an instance of the Traveling Salesman Problem and provides links to relevant resources. *Any* correct instance of TSP would have worked equally well. Furthermore, the programmer could have expressed the graph as an adjacency matrix, or could have mapped the graph to a list of the nodes traversed, the subgraph that makes up the tour, etc.

AλgoVista is based on a technique known as *program checking* [4] developed in the theory community as an alternative to testing and proofs of correctness. The idea is for a programmer to extend their programs with *checkers* to allow them to verify the correctness of the results they compute. AλgoVista's database of problem descriptions is, in fact, populated with such checkers, called *checklets*.

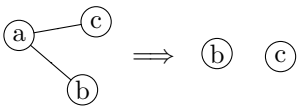
AλgoVista has many uses in an educational setting [8] and we will explore some of them in this paper:

1. Students can test their own understanding of a particular algorithm by asking AλgoVista if a particular drawing represents an instance of a particular problem. For example, a student could draw the the query



   to ask "is the path on the right of ==> a *traveling salesman tour* of the graph on the left?"

2. Students involved with algorithms-related research projects will often find that they are looking for an algorithm for a problem that they have not covered in any of their classes. AλgoVista provides a simple way of searching for and classifying such problems. For example, a student could draw the query



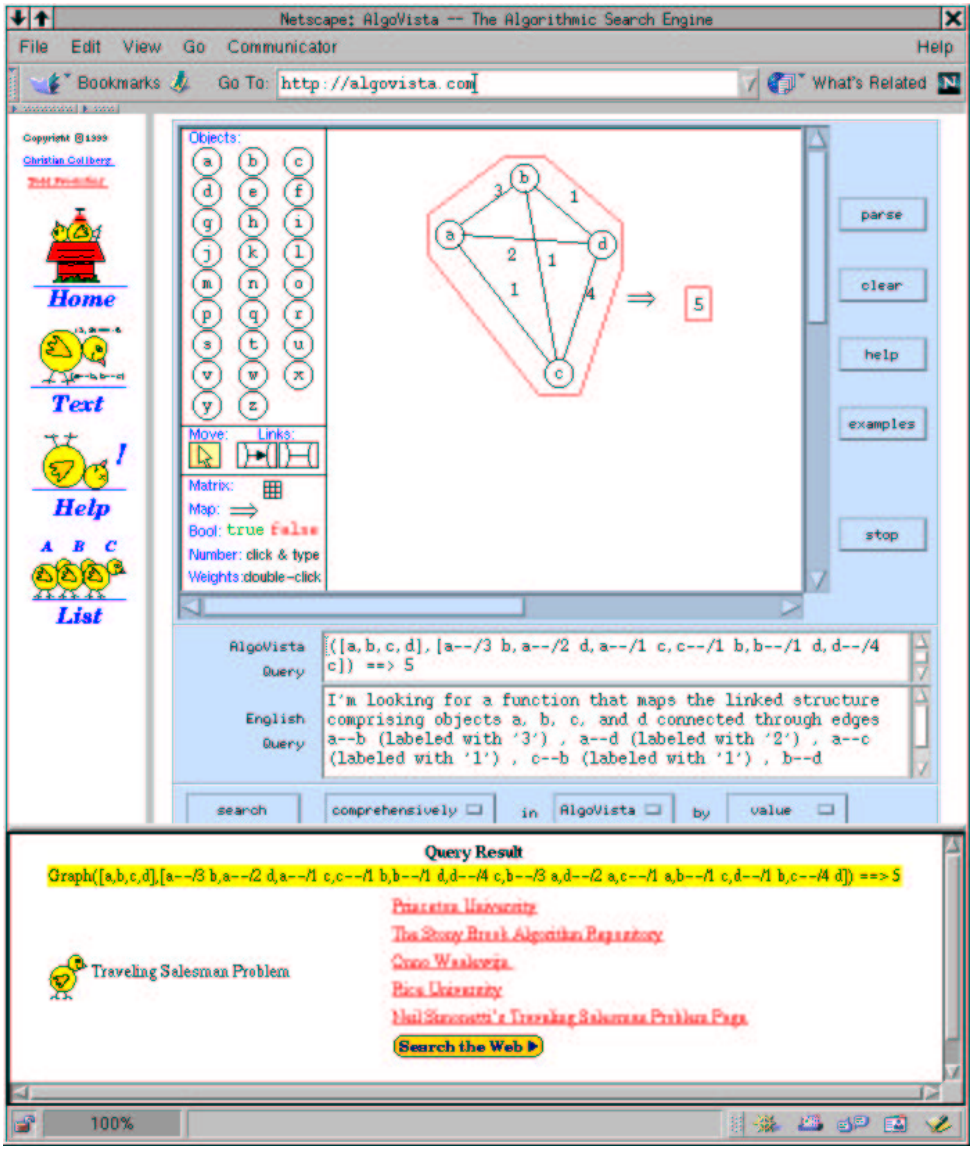
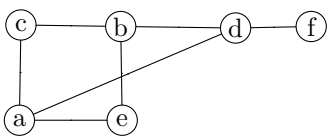


Figure 1: The AλgoVista user interface. The top right frame is an applet into which the user can enter a graphical query. The applet also displays the textual and English language queries corresponding to the graphical one. The bottom frame shows the result of the search.

---

which asks "what is the name of a problem that takes the graph on the left as input and produces the list of nodes on the right". AλgoVista would answer "this could be an instance of the *maximum independent set problem*," and give links to more information about this problem.

3. Students can use AλgoVista to classify combinatorial structures that they encounter in their studies. For example, a student can draw the query



to ask "what kind of graph is this" to which AλgoVista would answer "this is a *bipartite graph*," and give links to more information about this structure.

4. Students wishing to check the correctness of their own programs can add calls in their code to the checkers in the AλgoVista database.

5. Finally, instructors can use AλgoVista to set novel and interesting assignments in classes on algorithms, databases, and geometry.

In the remainder of this paper we will discuss how AλgoVista was used in two classes at the University of Arizona, an undergraduate class on algorithms and data-structures and a graduate theory of computation class.

2. Our Contribution

Coming up with novel assignments in algorithms and data-structure classes is always challenging. Often, programming assignments will ask the students to implement a well-known algorithm (such as Dijkstra's Shortest Path algorithm) or data-structure (such as a Red-Black Tree). Unfortunately, students can easily "borrow" solutions off the web for such problems; a simple Google query `"Dijkstra's Algorithm Java"` will lead directly to a solution in source code.

In two classes at the University of Arizona we instead asked students to write *checklets* for AλgoVista. This turned out to have several advantages:

- A unique aspect of AλgoVista is that it is a "communal" database, which means that anyone can upload new specifications into the system. By extending AλgoVista, students can feel a sense of "lasting accomplishment"; they have added to a knowledge database that others one day may use. This is in contrast to a standard assignment which is thrown away once it has been graded.

- Checklets are interesting to write. For example, a checklet for the Topological Sorting problem is similar to, but yet very different from, the Topological Sorting algorithm given in an algorithms textbook.[1] Thus, students are required to come up with their own algorithms rather than copying them from their textbook.

- A checklet is an executable specification of a particular problem. However, no special specification language need to be learned since checklets are written entirely in Java which most students already know.

- AλgoVista is a completely self-contained web-based system. Users do not need to download or install any special software. Rather, they upload new problem specifications by filling out forms in a standard web-browser.

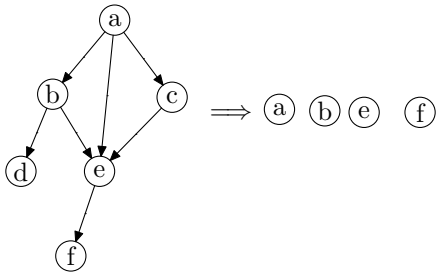
AλgoVista was used for programming assignments in two different classes during the Spring of 2002. "Foundations of Computing", CS344, is an undergraduate course focusing on the use of mathematical induction in problem solving, in program verification, and in proofs. An assignment required students to write checklets for the following graph algorithms and properties: Eulerian cycle, Hamiltonian cycle, testing if a given graph is connected, and testing if a given graph is a tree. A second class, "Theory of Computation" CS573, is a graduate course in complexity theory, approximation and randomized algorithms. One of the homework assignments required students to write checklets for the NP-complete problems Integer Programming, Longest Path, Set Partition, Max-Cut, and Dominating Set.

3. Example Assignment

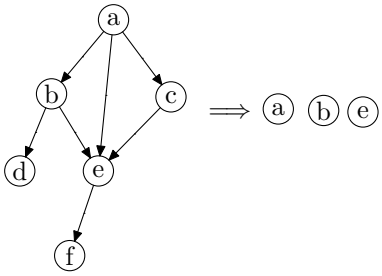
Here is a description of an assignment using AλgoVista. The students are given a brief summary about AλgoVista and encouraged to try out examples that are already in the database. An example checklet is presented in detail as follows: Figure 2 shows a very simple class that checks for the rational addition problem. When you write a checklet you have to implement four methods. *Description()* simply returns a short string that describes the problem. *ProtoExamples()* returns a set of examples in a very simple query language. In the example in Figure 2, the query `⌜((4,7),(5,6))==>(59,42)⌝` says that the result of $\frac{4}{7} + \frac{5}{6}$ is $\frac{59}{42}$. The `References()` methods returns an array of web resources for the problem. Finally, the `Check()` method returns `true` if `⌜((a1,a2),(b1,b2))==>(o1,o2)⌝`. In this case, the checklet is said to *accept*, otherwise it *fails*. The arguments to `Check` can vary depending on what the structure of the query is that it checks for.

Typically, the assignment would require the students to create several checklets and to add them to the database. We consider one problem (the longest path problem) in detail below:

A checklet for the longest path problem should, for example, accept for the following query:



but fail (return false) for this one:



There are a couple of things to note about this query:

- There is more than one possible longest path, and the checklet should accept for all of them.

- Many other queries for the longest path problem would also be possible. For example, the user might choose to map the graph to the actual length of the longest path (4 in our case). Typically, this will mean that we have to write more than one checklet. This is not necessary for the current assignment.

- The user might have chosen to represent the graph as an adjacency matrix rather than a linked structure. AλgoVista will automatically transform a query between these common representations, so this is something you don't have to worry about for this assignment.

- The graphical query above can also be expressed textually:

```
Digraph([d,b,f,e,a,c],[b->d,a->b,b->e,
  e->f,a->e,c->e,a->c]) ==> [a, b, e, f]
```

  The first part of this query is the graph (expressed as a list of nodes followed by a list of edges), the second part is `==>` which separates input from output, and the last part is the list of nodes (the longest path of the graph). The AλgoVista applet actually translates the graphical query to this textual counterpart and transfers it to the search engine itself for processing.

[1] If the checklet computes a list sorted in topological order and compares it with the input instance, this will not always work correctly, since there could be different lists that are topologically sorted.

4. Example Implementation

Students are given detailed instructions how to write and add the checklet to the database:

1. First a student must go to the appropriate web page, `algovista.cs.arizona.edu`.
2. Next the student must register in order to contribute. This is not used to track the students but rather allows them to keep their own set of checklets separate from everyone else.
3. A template for the new checklet can be downloaded from the web page.
4. The student enters the checklet name, such as `LongestPath` and enters a query example, such as

   ```
   Digraph([d,b,f,e,a,c],[b->d,a->b,b->e,
      e->f,a->e,c->e,a->c]) ==> [a, b, e, f]
   ```

   for the longest path problem.
5. This results in a template class which is produced. The student can now modify and upload to the server. The template is pure Java — the student can do anything that they typically do when they write Java programs.

   Figure 3 shows the template for the longest path. The student's task is to replace the `return false` statement with code that determines if `olist` contains the nodes of the longest path of the input graph. It is OK for the checklet to throw an exception if the input is malformed in some way. As far as AλgoVista is concerned, throwing an exception is identical to returning `false` (i.e., not accepting).

   For NP-hard problems like Longest Path the students are allowed to use an exact (exponential time) or an approximation algorithm, since the instances are fairly small, and running time is not the main priority[2]. For problems in P, efficiency is required or else the entire database becomes slower.

   Every query is an instance of the class `algovista.server.CL.Object`. The API for AλgoVista contains many subclasses of `algovista.server.CL.Object` which are used to create queries on graphs, trees, vectors, etc.
6. The student is also responsible for searching the web to locate interesting information about the Longest Path problem. These links are then added to the `References()` method.
7. The student then uploads the checklet to the server for testing and compilation. If the AλgoVista server detects any errors in the checklet these are returned to the student who will get the opportunity to correct them, upload again, etc.
8. After the checklet has been successfully added to the database the student can immediately try some queries on it. The AλgoVista applet can be instructed to only search the user's checklets, rather than all the ones in the database. This is useful during debugging.

[2]Note that an optimization formulation of an NP-hard problem such as TSP is different from the decision problem normally found in textbooks. In the decision problem $\mathrm{TSP}_d = \langle G, k, T\rangle$ we are asked to verify that the tour T is a hamiltonian cycle of G with cost of most k. This can certainly be done in polynomial time. In the optimization problem $\mathrm{TSP}_o = \langle G, T\rangle$, on the other hand, we are asked to verify that T is a minimum cost tour of G, which is hard.

5. Related Work

Interactive World Wide Web (WWW)-based learning tools are in the focus of research for a large number of computer science educators [5, 7]. Such tools allow students to actively engage in the learning process in and out of the classroom. Many of these, however, quickly become obsolete as hardware/software platforms and operating systems change. With the advent of platform-independent applications, there are far greater possibilities for creating more useful educational tools. While many computer science courses offer online access to handouts, syllabi, homework assignments solutions and other static documents, only a few have begun to exploit the full potential of the new technology available to us.

Intelligent tutoring systems have been an area of research in artificial intelligence for several decades. The primary application of these systems is providing feedback and tutoring the student rather than grading responses. Many systems, such as the LISP Tutor [2] and the Geometry Tutor [1], focus on abstract problem-solving skills and thus are more complicated than our aims here.

Since our notion of answer specification and checking involves a strong visualization component, it is also related to previous work on the visualization of algorithms and data structures. There is a rich literature that describes the benefits of concept visualization in education settings. Algorithm animation is a standard tool to allow students to visualize an otherwise abstract concept. Algorithm animation has been successfully used for visualizing graph algorithms, sorting, and searching, to name a few examples [11]. Similarly, program code animation also helps in the learning of new programming languages. Finally, concept animation has also been successful in communicating difficult concepts such as finite state automata [6]. Tools for creating animations of data structures and algorithms have also been developed [10]. Interactive tutorials have been designed and their positive impact on student learning evaluated [3]. Electronic books have been proposed and developed, in which hypertext, interactive animations, audio and video parts are integrated in a web-based stand-alone educational resource [5].

6. Summary and Future Work

We have shown how the web-based, communal, AλgoVista database of executable problem descriptions can be used to create interesting programming assignments in theory and algorithms classes. The system was used for the first time during the Spring of 2002. We will continue to use it in classes during the fall semester to gather further experiences.

AλgoVista could also be used as a tool for program verification in a Software Engineering or mid-level programming class. Since the checklets in AλgoVista's database are accessible over the network they can be referenced by any student's Java program to check the correctness of their algorithm implementation. For example, a student writing a sorting routine could add the code below to their program to check that it has the correct run-time behavior:

```
public static int[] sort (int[] input) {
   int[] output = ...;
   if (!AlgoVista.check(input,output,"Sorting"))
       System.err.println("Sorting failed!");
}
```

Instructors could use AλgoVista checklets in a similar way to verify programming assignments. We hope to try these applications of AλgoVista in future classes.

AλgoVista is operational at `algovista.cs.arizona.edu`. Any educational institution can use it immediately or can choose to download the free source and install it on a local server. The system is described in detail in [9].

7. REFERENCES

```
public class RationalAdd extends algovista.server.database.IntPair_IntPair2IntPair {

   // A short description of the Rational Number Addition problem.
   public String Description() {return "Addition of Rational Numbers";}

   // One or more accepting examples. These are used during checklet verification.
   public String[] ProtoExamples() {return new String[]{"((4,7),(5,6))==>(59,42)"}; }

   // References (consisting of a name and a url) to online resources.
   public algovista.server.database.Reference[] References() {
      return new algovista.server.database.Reference[]{
               new algovista.server.database.Reference(
                 "Ask Dr. Math",
                 "http://forum.swarthmore.edu/dr.math/faq/faq.integers.html")
             };
   }

   // The checker for the problem. Returns true if the arguments form an instance of
   // of the Rational Number Addition problem. Returns false or throws an exception otherwise.
   public boolean Check(long a1,long a2,long b1,long b2,long o1,long o2) throws Throwable {
       a1 *= b2 * o2;
       b1 *= a2 * o2;
       o1 *= a2 * b2;
       return ((a1 + b1) == o1);
   }
}
```

Figure 2: A simple checklet.

```
public class LongestPath extends algovista.server.database.Digraph2Vector {
   public String[] ProtoExamples () {
      return new String[]{
                  "Digraph([d,b,f,e,a,c],[b->d,a->b,b->e,e->f,a->e,c->e,a->c])==>[a,b,e,f]"};
   }
   public boolean Check (
      algovista.server.CL.linked.Digraph in_G,
      algovista.server.CL.Object olist[]) throws Throwable {
      return false;
   }
}
```

Figure 3: A checklet template for the Longest Path problem. `in_G` **is the input graph and** `olist` **the list of nodes in the (proposed) longest path.**

[1] J. R. Anderson, C. F. Boyle, and G. Yost. The geometry tutor. In A. Joshi, editor, *Proceedings of the 9th International Joint Conference on Artificial Intelligence*, pages 1–7, Los Angeles, CA, Aug. 1985. Morgan Kaufmann.

[2] J. R. Anderson and B. J. Reiser. The LISP tutor: It approaches the effectiveness of a human tutor. *Byte Magazine*, 10(4):159–175, Apr. 1985.

[3] L. Barnett, J. Casp, D. Green, and J. Kent. Design and implementation of an interactive tutorial framework. In *Proc. 29th SIGCSE Tech. Symp.*, pages 87–91, 1998.

[4] M. Blum and S. Kannan. Designing programs that check their work. *Journal of the Association for Computing Machinery*, 42(1):269–291, Jan. 1995.

[5] Boroni, Goosey, Grinder, Lambert, and Ross. Tying it all together: Creating self-contained, animated, interactive, web-based resources for computer science education. *SIGCSEB: SIGCSE Bulletin (ACM Special Interest Group on Computer Science Education)*, 31, 1999.

[6] C. Boroni, F. Goosey, M. Grinder, and R. Ross. Weblab! A universal and interactive teaching, learning, and laboratory environment for the World Wide Web. In *Proc. 28th SIGCSE Tech. Symp.*, pages 199–203, 1997.

[7] C. M. Boroni, F. W. Goosey, M. T. Grinder, and R. J. Ross. A paradigm shift! the Internet, the Web, browsers, Java and the future of computer science education. In *Proceedings of the 29th SIGCSE Technical Symposium on Computer Science Education (SIGCSE-98)*, volume 30(1) of *SIGCSE Bulletin*, pages 145–152, Feb. 25–Mar. 1 1998.

[8] C. Collberg, S. G. Kobourov, S. Westbtook, and J. Miller. Algovista: A tool to enhance algorithm design and understanding. In *7th Ann. Symp. on Innovation and Technology in Comp. Sci. Education*, page 253, 2002.

[9] C. S. Collberg and T. A. Proebsting. Problem classification using program checking. In *2nd Int'l Conference on FUN with Algorithms*, pages 61–83, 2001. `http://www.cs.arizona.edu/~collberg/FUN.ps.`

[10] W. Pierson and S. Rodger. Web-based animation of data structures using JAWAA. In *Proc. 29th SIGCSE Tech. Symp.*, pages 267–271, 1998.

[11] J. Stasko, J. Domingue, M. H. Brown, and B. A. Price, editors. *Software Visualization: Programming as a Multimedia Experience*. MIT Press, 1998.