In *Proceedings of the Fifth International Workshop on Artificial Intelligence and Statistics, 1995.*

Tools for Empirically Analyzing AI Programs

Scott D. Anderson, David M. Hart,
David L. Westbrook, Paul R. Cohen
and Adam Carlson

Computer Science Technical Report 94-78

Experimental Knowledge Systems Laboratory
Computer Science Department, Box 34610
Lederle Graduate Research Center
University of Massachusetts
Amherst, MA 01003-4610

Abstract

The paper describes two separate but synergistic tools for running experiments on large Lisp systems such as Artificial Intelligence planning systems, by which we mean systems that produce plans and execute them in some kind of simulator. The first tool, called CLIP (Common Lisp Instrumentation Package), allows the researcher to define and run experiments, including experimental conditions (parameter values of the planner or simulator) and data to be collected. The data are written out to data files that can be analyzed by statistics software. The second tool, called CLASP (Common Lisp Analytical Statistics Package), allows the researcher to analyze data from experiments by using graphics, statistical tests, and various kinds of data manipulation. CLASP has a graphical user interface (using CLIM, the Common Lisp Interface Manager, Version 2.0) and also allows data to be directly processed by Lisp functions. CLIP and CLASP form the foundation of a larger set of specialized tools we are building for the empirical analysis of AI programs.

This research is supported by ARPA/Rome Laboratory under contract #'s F30602-91-C-0076 and F30602-93-C-0100.

1 Introduction

As planning problems become more complex, involving hundreds of objects and thousands of resources (e.g., ships, planes, trucks, satellites), researchers will need to turn to simulators, controlled experiments, and statistics to study the behavior of their systems. In this paper we will introduce two tools that we have developed to aid in running and analyzing experiments: CLIP and CLASP (Common Lisp Instrumentation Package and Common Lisp Analytical Statistics Package). These tools are described in more detail in Anderson et al. ([2]), where we give examples of their use with a planning system for a transportation planning problem. They form the substrate for a larger toolbox we are developing that is specifically designed for analyzing AI programs.

CLIP enables researchers to define experiments in terms of the conditions under which the simulator is to be run and data collected. CLIP also helps with the running of the experiment, by looping over all the experimental conditions, running the simulator, and writing the data to files. At that point, a researcher will want to analyze the data using statistical software. While the data files that CLIP writes can be analyzed by any statistical package, CLIP is especially well integrated with CLASP. CLASP has many of the standard descriptive and inferential statistics, together with a convenient graphical user interface, and a Lisp interaction window that researchers can use for implementing statistical operations that we have not anticipated. CLIP and CLASP were developed to be portable across Unix Common Lisp platforms. [1]

2 Running Experiments

A great many experiment designs are used in science, but most of them can be viewed as sets of *trials*, each with a number of independent variables, representing the conditions under which the trial is run, and a number of dependent variables, which are the objects of scientific scrutiny. This is the simplest of the kinds of experiment designs that CLIP supports. One common kind of experiment within this paradigm is called a "fully factorial" design, in which there are one or more *factors*, each of which has a small number of discrete levels. Another common kind of experiment looks at the relationship of two or more continuous variables, such as the correlation between them. CLIP supports both of these experiment designs.

2.1 Instrumentation

Adding code to extract information from a system is called *instrumentation*, hence CLIP's name. Most of CLIP's functionality is directed towards extracting different kinds of information from your system—information that is calculated afterwards, collected periodically during execution, or possibly collected whenever some event occurs. This aspect of CLIP is deferred to section 2.2. First, we present an overview of how CLIP works and what you need to do to use it.[2]

To use CLIP to run an experiment, CLIP first needs to know how to run your simulator. Essentially, this is a single function or piece of code that CLIP can call to start a trial and which will return when the trial is over. CLIP also works with simulators that run in multi-threaded (multiple process) Lisps, but it nevertheless treats the simulator as a single piece of code.[3] Between trials, CLIP will need to reset your system, although this might be unnecessary if the simulator is purely functional (few are). If your simulator has a notion of time, such as having a clock, and you want CLIP to schedule events for particular times, CLIP will need to know how to interact with the scheduler and the clock. For example, you might want to collect data every day of the simulation, with the average being

[1] For specifics about the workstations and Common Lisp implementations that are supported, see the Release Notes for CLIP and for CLASP available at the ftp site listed in section 6

[2] This article is no substitute for the CLIP/CLASP manual [1], where everything is rigorously explained.

[3] This requirement may be lifted in future versions of CLIP, but the impact is minor. Most multi-threaded Lisps provide a `process-wait` function, which can be used to make the simulator seem like a single piece of code.

written to the data file. To describe how to run and control your simulator, there is a single CLIP macro, called `define-simulator`.

Next, you will define your experiment, which is again done with a single CLIP macro, called `define-experiment`. The heart of an experiment is the set of independent and dependent variables, which are specified with the macro. The independent variables are described with a simple syntax much like the Common Lisp `loop` macro. The names of dependent variables are simply listed; the definition of how to collect and report the data is separately defined as objects called "clips," which will be discussed in the next subsection. The `define-experiment` macro also provides ways for the user to run code during the experiment, at four distinct times:

Before the Experiment: When the experiment gets started, you may want, for example, to load special knowledge-bases or set scenario parameters. This is also a chance to do more mundane things, such as allocating data structures or turning off the screen-saver.

Before Each Trial: At each trial, you may want to reinitialize parameters and data structures. One important thing to do is to configure your simulator for the current experimental condition.

After Each Trial: The most important thing that is typically done after each trial is to call the function `write-current-experiment-data`, the CLIP function that writes all the data for this trial. This is also a good time to run the garbage collector.

After the Experiment: Typically, code run after the experiment undoes the code run before the experiment, such as deleting data structures or turning on the screen saver.

Of course, any arbitrary code can be executed at these times, for whatever purposes you want. The key idea is that the before- and after-trial code surrounds every trial and runs many times, while the before- and after-experiment code surrounds the whole experiment and runs only once. This ability to run arbitrary code is more than just an opportunity for hacks—it is a clear and precise record of the exact experimental conditions. Records are important as a memory aid and as a means for replicating experiments.

When the experiment has been defined, you start it running with the function `run-experiment`. This function takes arguments, which you can refer to in the before/after code, so that the final specification of the experimental conditions can be deferred until run-time. The `run-experiment` function also allows you to specify the output file for the data, the number of trials, the length of the trial, and other such information.

Defining the simulator and the experiment, and then running the experiment is fairly straightforward and is only a fraction of what must be done to run an experiment. The bulk of the effort is in defining "clips"—functions that measure the dependent variables of your experiment. Fortunately, they are modular and reusable.

2.2 Clips

Clips are named by analogy with the "alligator clips" that connect diagnostic meters to electrical devices. They measure and record aspects of your system (the values need not be numerical). Essentially, they are Lisp functions that you define and which CLIP runs if they are included in the definition of the experiment. Once written, they can be mentioned in any number of experiments. Indeed, it is common to build up files of clips, so that a new experiment can be quickly defined by writing a `define-experiment` form (or editing an old one) and listing the clips in the `instrumentation` argument to `define-experiment`.

Clips are defined with the `defclip` macro, which is very much like `defun`, except that information added before the body is read by CLIP. The central issue in defining a clip is the time that it is run. (The code that is run is written in the `defclip` body and is entirely up to the user.) Most clips simply measure values after a trial is finished, for variables such as "finish date,"

"number of bottlenecks," and "total waiting time for ships." More complicated clips may need to run periodically, which only makes sense for simulators that have a clock of some sort; CLIP will schedule the clip using the `schedule-function` specified in the `define-simulator` form. Other clips may need to run when some event occurs; this is accomplished by tying the clip to a function in your simulator, using a mechanism like the "advise" facility found in many Lisp implementations. The `defclip` form has syntax for tying the clip to a function. When a clip is run many times during a trial, it can either report the mean of the values or it can report all the values (or some function of them), as *time series* data. We can statistically analyze time-series data to see if there are temporal correlations. We cannot answer such questions just by looking at mean values after a trial is over.

CLIP has other features to support experimentation, such as aborting a trial but continuing the experiment, say when some intermittent error has occurred—very common in stochastic simulations. CLIP also lets you run only part of the experiment, which facilitates breaking the experiment into parts to run on different machines. These facilities are all explained at length in the CLIP/CLASP documentation [1].

3 Data Analysis

The idea of CLASP began when we wanted to run a t-test on some experiment data without having to write out the data to a file in some tab-delimited format, move the code to another machine, run a statistics program, and load the data. From this small beginning, we have added most of the workhorse statistical functions, data manipulation (regrouping, selecting subsets), data transformation (such as log transforms), graphing software (now replaced by SCIGRAPH, by Bolt, Beranek and Newman, Inc.). We have a convenient graphical user interface implemented in CLIM, and a programmatic interface so that the CLASP functions can be called by the user if the desired data manipulation isn't already on a menu. Ideally, everything can be accomplished by menus in the graphical user interface.

CLASP's screen interface, an example of which is shown in figure 1, comprises four areas: the menus, the datasets, the results, and the notebook:

Menus The CLASP menus will appear across the top of the window. The menus, which will be discussed below, are: File, Graph, Describe, Test, Manipulate, Transform and Sample.

Datasets When you load a file of data into CLASP, such as a file written by CLIP, it becomes a CLASP *dataset* and appears on this menu. The name of the dataset is the name of the experiment. Each column of data is called a *variable*; the name of the variable is usually the name of the clip that returned that variable, unless you specify a different name in the `defclip`. Most operations in CLASP take either datasets or variables as arguments, and the items in this pane become mouse-sensitive under those circumstances.

Results Display When a CLASP operation yields a complex result, such as a table or graph, that object goes into a menu of results.

Notebook The notebook is a complete Lisp read-eval-print loop, except that CLASP commands are also accepted. Having Lisp available is important and powerful, because users can operate on the data in ways we have not yet implemented or even thought of. CLASP commands can be typed instead of using the menus; indeed the menus just type the appropriate thing into the notebook. When the command is fully entered, it is executed and its results are printed to the notebook. CLASP output in the notebook is also mouse-sensitive when appropriate.

CLASP uses a prefix command syntax, very much like Lisp, in that you give the command

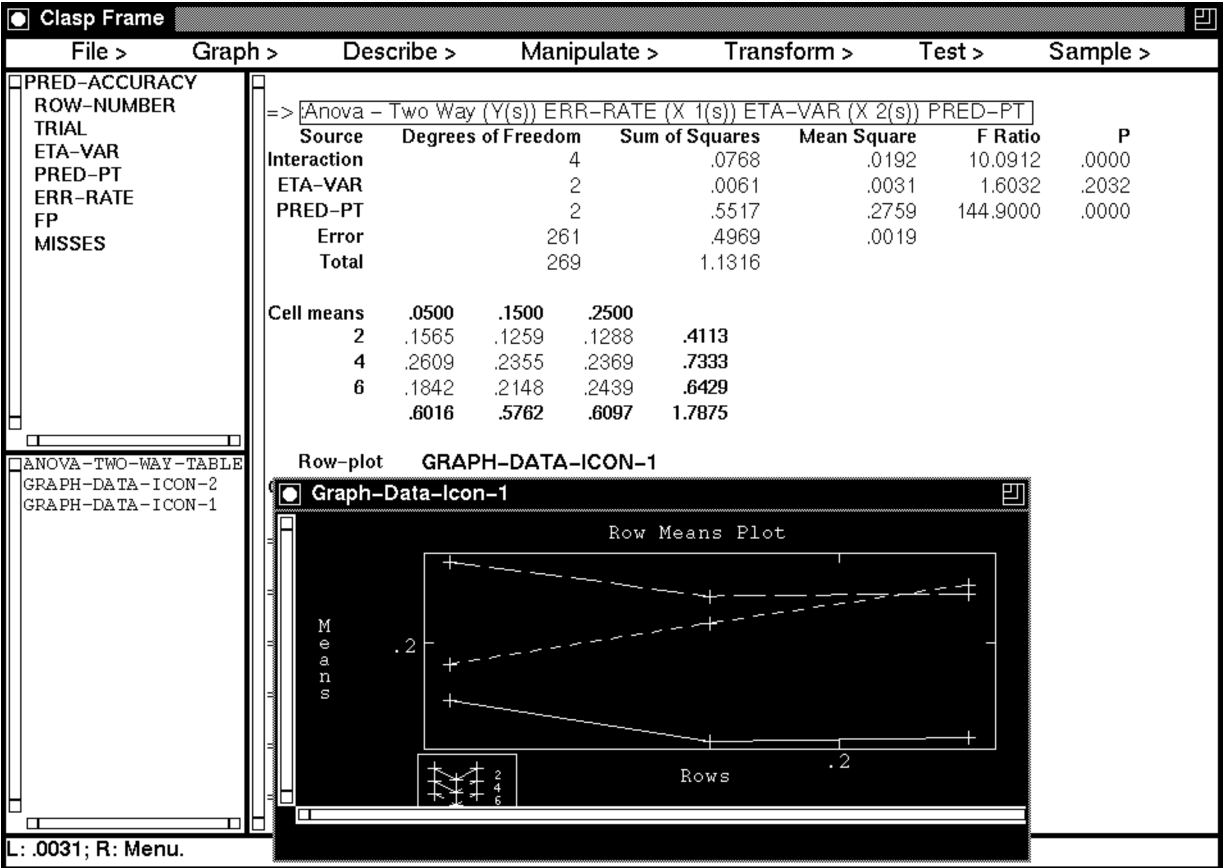

Figure 1: Excerpt from sample interaction with CLASP

name first, such as `:T Test Two Samples` X Y, where X and Y are variables. Using the features of CLIM, CLASP allows command completion and prompts for arguments.

CLASP groups related commands in the main menu. The following are the categories and the kinds of commands found in each. This description is just a few highlights, but everything is completely described in the CLIP/CLASP manual.

File This menu allows you to load CLASP datasets from files and to save them to files, say if you've made changes or created new datasets.

Graph This menu allows a number of displays of data, including histograms, scatter plots, line plots, and regression plots.

Describe This menu contains the most commonly used descriptive statistics.

Test This menu contains a variety of inferential procedures, including t-test, confidence intervals, analysis of variance, chi-square and regression. In the near future, we plan to implement bootstrap variants on most common statistical functions [7].

Manipulate CLASP provides several ways to extract subsets from a dataset through data manipulation operations such as partitioning. Other operations on this menu allow you to create new datasets.

Transform This menu has commands that produce new variables from old ones (for example, by sorting a variable).

Sample This menu contains commands that produce datasets by sampling from a given

probability distribution.

4 Empirical Analysis Toolbox

CLIP/CLASP forms the core of a larger set of empirical tools for analyzing the behavior of AI programs that we are building. These tools are "add-on" modules to CLIP/CLASP that help the user find significant relationships in data and model the causes of these relationships. These modules are more focused than CLASP's general statistical procedures. Each is tailored to a specific aspect of program analysis, such as finding the major factors contributing to program success or identifying interactions of program components that degrade performance. The modules we are currently building, while by no means complete, include three that have proved particularly useful:

Exploratory Data Analysis Having run an experiment and gathered data, the user is faced with the task of identifying significant relationships among the factors measured. We are building a module that assists the user in this effort by employing EDA techniques [11]. These techniques can partition data to distinguish different modes of behavior and generate functional descriptions of interactions between factors. Through detailed exploration of experimental data the user can gain a more complete picture of system behavior.

Dependency Detection The complexity of AI programs has reached a point where their behavior can be difficult to predict and problems difficult to replicate. Program actions often interact in unforeseen and deleterious ways. We employ a technique we call *dependency detection*, analyzing program execution traces with a statistical filter to find significant dependencies among interacting actions [8, 9, 10].

Causal Induction Having explored the data and/or identified dependencies among interacting factors, the user next tries to build a predictive model of the program's behavior. We would like for such a model to tell us how to change the program to improve or modify its behavior. This requires that we understand the underlying causal relationships among the factors influencing its behavior. We are developing a module that uses *path analytic* techniques to build causal models from data [4], and have incorporated into it several new *causal induction* algorithms [6].

Case studies using these and other empirical techniques to analyze AI programs are included in a forthcoming textbook on empirical methods for AI research [5]. It is possible that the major contribution of CLIP/CLASP will not be as a standalone instrumentation and analysis package, but rather as a platform for the integration of more powerful techniques such as those described above. We envision a new generation of statistical software in which knowledge and heuristics will guide the application of exploratory data analysis procedures. We are currently at work on such a system which we call AIDE, automated intelligent data exploration [11].

5 Related Work

An alternative to CLIP is the METERS system, developed by Bolt, Beranek and Newman, Inc., for use in the ARPA/RL Planning Initiative's Common Prototyping Environment [3]. METERS is particularly useful for collecting and filtering time-series data from distributed systems. XLISPSTAT [12] provides a richer set of statistical and graphical capabilities than the current version of CLASP, but is not as tightly integrated into an environment for instrumentation and analysis as is CLASP. CLASP's CLIM interface also makes it more portable across the numerous Common Lisp platforms.

6 Current Status

CLIP and CLASP are included as evaluation tools in the ARPA/RL Planning Initiative's Common

Prototyping Environment and will soon be incorporated into Rome Laboratory's Advanced AI Technology Testbed. CLIP/CLASP is being used to instrument and analyze AI systems for planning, scheduling, causal induction, molecular biology, signal interpretation, and others.

A Macintosh version of CLASP, MACCLASP, is currently being prepared for distribution with the textbook mentioned above. Examples in the text are all analyzed with MACCLASP. MACCLASP's interface is implemented using the Mac toolbox (instead of CLIM – the MCL version of CLIM has some quirks that make the original CLASP interface difficult to use). MACCLASP benefits from the intuitive look and feel of the native Macintosh interface, though it lacks some important features of the CLIM version, notably the integrated Lisp Listener and convenient logging capabilities.

CLIP and CLASP may be obtained by anonymous ftp from `ftp.cs.umass.edu`. CLIP can be found under the directory `pub/eksl/clip`, CLASP under `pub/eksl/clasp`; manuals and information about which platforms are supported are included in both these directories. A tutorial on CLASP is available under `pub/eksl/clasp-tutorial`. For more information about MACCLASP contact `clasp-support@cs.umass.edu`.

Development of CLIP/CLASP continues, and is largely driven by user demand. We will continue to add useful statistical tests and data manipulation functions, as well as useful functions contributed by the user base. Comments, bugs, new feature requests and general questions can be sent to `clasp-support@cs.umass.edu`.

7 Conclusion

CLIP works directly with a user's simulator, helping the experimenter define dependent measures, control independent variables and run experiments. CLASP is a statistics package and as such competes with many good statistics packages on the market. Its advantages are that it is implemented in Common Lisp and CLIM, so that it can easily be combined with your simulator and with CLIP, allowing for a completely integrated experimental environment. These tools form the nucleus of an extended toolbox for the empirical analysis of AI programs we are developing. We believe such support for empirical science will be of significant benefit to the AI community.

Acknowledgements

This research is supported by ARPA/Rome Laboratory under contracts #F30602-91-C-0076 and #F30602-93-C-0100. The US Government is authorized to reproduce and distribute reprints for governmental purposes notwithstanding any copyright notation hereon. We thank the many current and former members of EKSL who helped develop CLIP and CLASP, and in particular Rob St. Amant, whose insights helped in the preparation of this paper.

References

[1] Scott D. Anderson, Adam Carlson, David L. Westbrook, David M. Hart, and Paul R. Cohen. CLASP/CLIP: Common Lisp Analytical Statistics Package/Common Lisp Instrumentation Package. Computer Science Dept. Technical Report 93-55, Univ. of Massachusetts/Amherst, 1993.

[2] Scott D. Anderson, Adam Carlson, David L. Westbrook, David M. Hart, and Paul R. Cohen. Tools for experiments in planning. To appear in *Proc. of the Tools with Artificial Intelligence Conference.* 1994.

[3] Bolt Beranek and Newman, Inc. and ISX Corporation. Common Prototyping Environment Testbed Release 1.0: User's Guide, 1993. BBN Systems and Technologies, 10 Moulton Street, Cambridge, MA 02138.

[4] Paul R. Cohen, Adam Carlson, Lisa Ballesteros and Robert St. Amant. Automating path analysis for building causal models from data. In *Proc. of the Tenth International Conference on Machine Learning.* Pp. 57-64. Morgan Kaufmann, 1993.

[5] Paul R. Cohen. *Empirical Methods in Artificial Intelligence.* MIT Press. Forthcoming.

[6] Paul R. Cohen, Dawn Gregory, Lisa Ballesteros and Robert St. Amant. Two algorithms for inducing structural equation models from data. To appear in *Proc. of the Fifth International Workshop on AI and Statistics.* 1995.

[7] Bradley Efron and Gail Gong. A leisurely look at the bootstrap, the jackknife, and cross-validation. *The American Statistician*, 37(1):36–48, February 1983.

[8] Adele E. Howe and Paul R. Cohen. Understanding planner behavior. To appear in *Artificial Intelligence.*

[9] Adele E. Howe. Finding dependencies in event streams using local search. To appear in *Proc. of the Fifth International Workshop on AI and Statistics.* 1995.

[10] Tim Oates, Dawn Gregory and Paul R. Cohen. Detecting complex dependencies in categorical data. To appear in *Proc. of the Fifth Intern. Workshop on AI and Statistics.* 1995.

[11] Robert St. Amant and Paul R. Cohen. Preliminary system design for an EDA assistant. To appear in *Proc. of the Fifth International Workshop on AI and Statistics.* 1995.

[12] Luke Tierney XLISPSTAT. School of Statistics Report #528, Univ. of Minnesota, 1988.