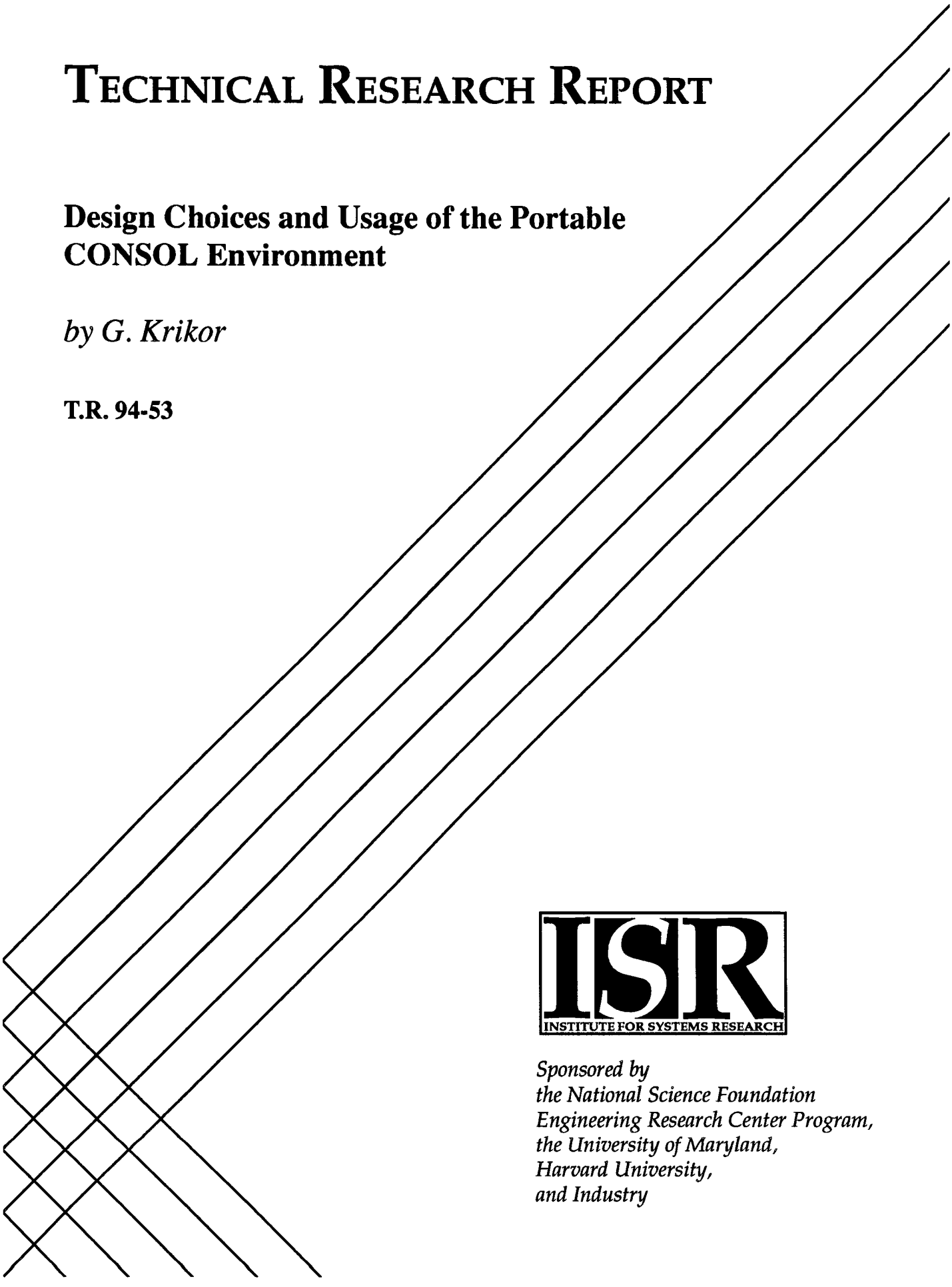
TECHNICAL RESEARCH REPORT

Design Choices and Usage of the Portable CONSOL Environment

by G. Krikor

T.R. 94-53



Sponsored by
the National Science Foundation
Engineering Research Center Program,
the University of Maryland,
Harvard University,
and Industry

Design Choices and Usage of the Portable CONSOL Environment

Gebran Krikor

June 3, 1994

Abstract: This document deals with the design choices and goals of the **Portable CONSOL** project. The original version of CONSOL-OPTCAD, a software package for the optimization-based design of engineering systems developed at the University of Maryland, was designed for use under SunOS on UNIX workstations. The idea behind the **Portable CONSOL** project is to create a portable engine which implements the functionality of CONSOL-OPTCAD but which will make versions of CONSOL for other computer platforms possible.

Given the technological advancements since the original implementation of CONSOL-OPTCAD, it is now feasible to have a version of CONSOL which can be run on many different computer platforms. With personal computers now available which are as powerful (or more powerful) than the UNIX workstations for which CONSOL-OPTCAD was originally designed, a version of CONSOL capable of running on desktop personal computers is now desirable. One of the results of this project has been the development of a version of CONSOL which can be run on multiple architectures, including SunOS, Linux, MS-DOS, MS-Windows, and OS/2.

Other results from this project include new user interfaces for CONSOL, which allow for easier user manipulation and control of the optimization process. Support for more external simulators is being developed, as well as new graphical tools to enhance the usefulness of CONSOL as an optimization tool.

Contents

1 Introduction

1.1 Background

CONSOL-OPTCAD, a software package for optimization-based design of engineering systems, at the University of Maryland at College Park, is a prime example of computers being used to simplify and structure the development of system designs and prototypes.

When a system is first being developed, the engineer must consider the structure of the system. The nature of the problem, as well as other constraints imposed by the particular situation, cause the first design choices to be made. For example, in the design of a band-pass filter for a radio receiver, many competing designs exist, each with its own particular tradeoffs. Some are more reliable than others, some have fewer parts required, but sacrifice noise immunity. Other designs have more and differing requirements.

After the initial structure for the system has been decided, various tradeoffs must be considered. Going back to the radio-receiver example, the resonant frequency is determined by the equation $f = (1/(L * C))^{1/2}$. This equation does not specify the values of L and C, only their relationship to one another. The choice of L and C is determined by other factors such as part availability, noise immunity, the "quality" of the circuit, etc.

CONSOL-OPTCAD is a tool for exploring just these tradeoffs. Originally designed as a tandem package, composed of two programs Convert and Solve, CONSOL-OPTCAD combines the features of a simulator with those of an optimizer into one package. CONSOL-OPTCAD also provides facilities to allow the use of external simulators to evaluate the design specifications to be optimized. Most problems of engineering interest involve the use of external simulators.

Problem definition is done using a special *Problem Description Language* in which the system to be optimized is specified in concrete terms. Such a description is referred to as a PDF, *Problem Description File*. Convert is then invoked on the PDF and produces a C language simulator for the problem, as well as a binary file describing the initial conditions and the limits for the variables.

Once Convert has been run, Solve is invoked. Solve is intended to be an interactive optimizer, allowing the operator full control over the problem at all stages in the optimization. Features include graphical displays of the problem at each stage in the optimization, as well as the ability to change and freeze variables at any stage in the optimization process.

1.2 Goals

With the success of CONSOL-OPTCAD on the SunOS (UNIX) platform, it became apparent that the users of CONSOL-OPTCAD would need support for other platforms as well. This lead to the development of a VMS port of CONSOL-OPTCAD, which had most of the functionality of the UNIX version, but was lacking support for graphics, and had limitations in several other areas.

This experience, coupled with the demand for a more accessible version of CONSOL-OPTCAD, lead to the "Portable CONSOL" project. The goal of this project was to develop a version of CONSOL which could be compiled and run on any platform with a minimum of experience and of new coding. The goal was to have a version of CONSOL for both the MS-DOS and Microsoft Windows environments which would share as much code as possible with the UNIX and VMS versions.

2 Portable CONSOL

2.1 Background

CONSOL is divided into several major subsystems. These systems are primarily

1. algorithms - this subsystem is intended to be the primary problem solving engine. At this time, FSQP (Feasible SQP) and Monte Carlo are the primary optimization engines. Gateways to access external simulators are available in the UNIX version and should soon be available in the portable CONSOL as well. External simulators include MATLAB and WATAND (a circuit simulator).

2. commands - this subsystem is intended to provide the primary text based user interface to the application.

3. graphics - this subsystem is intended to provide a "platform-independent" graphics engine for providing plots and graphs involved with CONSOL.

4. lexical - this subsystem is intended to provide all of the lexical analysis needed for CONSOL. This includes parsing and identifying tokens, etc.

5. main - this subsystem is the main control for CONSOL. It handles initialization, configuration, etc. It also manages command input, etc.

6. others - this subsystem contains many of the common routines used throughout CONSOL. For example, a memory allocation routine which checks for error conditions, a file open routine which checks for error conditions, etc.

The primary goal of this subsystem is to provide consistent error checking and handling.

7. specs - this subsystem provides routines for the evaluation and determination of those quantities in need of optimization. It is also used by the "pcomb" command to determine the phase of the optimization.

2.2 Preliminary steps – preparation

The first task for the *Portable CONSOL* project was the updating of all of the source code to ANSI C/C++ standards. (The C++ standard being a "stricter" standard, it was decided to make the code conform to C++ standards). This meant developing functional prototypes for every function in CONSOL, and converting the function definitions from the K&R definition style to the ANSI definition style.

Also to be done was the removal of the majority of the cross-subsystem function calls. These calls were primarily from the *algorithm* subsystem to the *commands* subsystem; however, these calls could be found in many places throughout the code. The new version was designed to have a controlled number of essential cross-subsystem calls. The reason for limiting these calls was that the commands subsystem is not portable from system to system. (For example, in a graphical environment, the text-based command environment cannot be used.)

Another convention in the code which had to be changed was the use of "goto" statements. These statements have come to be regarded as statements which should be used only as a last resort. This is because they normally lead to code which can be very difficult to follow. These statements were used in many places in the code for a variety of reasons (handling error conditions, etc). They were most heavily used in the *commands* section of the code and could be found in almost every function in that section.

Lastly, the code was heavily machine dependent. It assumed that the size of integer variables is equal to that of pointer variables which is equal to that of floating point variables, etc. This is not true in many computer architectures, and in fact, it is only due to the peculiarities of the SunOS C compiler, as well as the "sun3" and "sun4" machine architectures, that these assumptions did not cause problems in the original versions of CONSOL.

2.3 Step 2, prototyping and machine independence

Making these changes to update the code revealed a large number of further problems with the source. The UNIX code had many cases where typechecking was not enforced in parameter passing for function calls. For example, a function

would be defined with two character type variables and would receive one integer variable when it was actually called. It was only because of the sun4 computer architecture that these discrepancies did not cause execution problems in the executable.

For example, a function defined by

```
void quit(int status, char *s1, char *s2);
```

would be called as simply

```
quit(0);
```

The two parameters s1 and s2 would be left as garbage on the stack, and on another system could (and eventually did) cause the executable to crash.

The cross-subsystem function calls were removed without incident. Initially, their removal caused a loss of functionality, but cleaner ways of achieving the same functionality were devised.

The goto statements, while inconvenient, were left in place, where they either handled somewhat exotic error conditions (the others subsystem), were in code that would eventually be completely rewritten (the commands subsystem), or actually caused the compiler to produce more efficient assembly language code on some platforms (in the algorithms subsystem).

The changes to the system to remove the machine dependencies were for the most part successful, although they did involve some major alterations to the algorithms subsystem. The changes involved in making the commands and others subsystems machine independent revealed some more prototyping errors which were quickly fixed.

2.4 Step 3, modularization

At this stage, code had been developed which would *compile* on several major platforms (SunOS, Linux, MS-DOS, MS-Windows, and OS/2); however, this code could not actually be executed on these platforms. There are several *operating system* specific issues which must be dealt with. The next stage was to isolate all of the *system* specific features of CONSOL and strictly modularize the code based upon those guidelines.

Analysis of the code at this stage revealed three major *system* specific features of CONSOL. These were

1. *Linking to native simulators:* Linking to PDF simulators was done "dynamically" under SunOS, using a feature of the operating system where an executable program (object file) could be loaded into memory, the executable "patched" into the running program (symbols resolved, etc), and simply called as a normal procedure call. A portable (or at least simple)

interface and method for performing such linking under new systems was required. Many of the potential computer platforms for CONSOL simply did not support (in any way) dynamic linking.

2. *Multitasking/multiprogramming considerations:* Under a system like MS-DOS, a single-tasking operating system, the question of installing and supporting external simulators becomes far more difficult. A consistent and easy-to-use interface for external simulators would need to be developed.

3. *I/O considerations:* Under an operating environment like MS-Windows, the standard text mode interface of CONSOL could not be used. MS-Windows, being a graphical environment, does not support (well) purely textual interfaces. Also, using a graphical environment would allow the use and development of a far more user-friendly environment. (For example, menus and button bars to allow access to commonly and frequently accessed functions.)

The first two of these requirements are purely programmer issues. The last requirement is a user-interface issue.

The need for a user-friendly environment lead to the consultation of the IBM and Microsoft standard *CUA* guide. Common User Access (CUA) is a standard published by IBM (and Microsoft, in the past) which details the specific guidelines to follow in designing user interfaces in the MS-Windows and OS/2 operating environments. The requirements of this system are very similar to those embodied in the Motif (X Window System) "look-and-feel" description. In addition, other sources were consulted in the hopes of developing a usable but fast interface. For example, portions of the user interface were patterned after those found in popular Windows programs. The most notable of these features is the MDI interface (Multiple Document Interface). Also of note is the "ButtonBar" which provides easy mouse access to important features.

With these resources, a preliminary user interface design was constructed. This design had to take into account the two major interfaces available on most platforms, text-based and graphical interfaces. Using this design, three interfaces were planned for implementation.

The first was a text-based user interface, almost identical to that of the original UNIX version. This interface would be used in text-based environments. It would be purely text input command driven, with possible support for graphical plots.

The second was a graphical user interface best suited to native graphical environments. It would include menus, multiple windows, graphics, listboxes, etc., all of which are the "staples" of a graphical environment.

The third was an attempt to develop a windowed text-based environment (similar in look-and-feel to the graphical environment), but the lack of a small, *freely available* text-windowing package made this option unfeasible, and it was dropped. Another reason for discarding the third interface was the fact that

most text-based systems possess a graphical environment (MS-DOS -> MS-Windows, OS/2 -> Presentation Manager, UNIX -> X11), so that if users require the features of CONSOL under a graphical environment, such an environment is normally available (or can be purchased) for that platform.

2.5 Final decisions

The final decision was to support both a text and a graphical interface, one for each type of environment. The text-based interface can be designed such that it will work with few to no changes when supporting a new platform. Unfortunately, the advantages of the ease of use of a graphical interface are offset by the relative programming complexity required to support each different environment.

3 Modules

3.1 Dynamic linking of native simulators

The code for doing linking to simulators was isolated into one group of functions in one source file. The syntax for the function was changed to support the various possible formats for dynamic linkers. For example, some work with the loading of an object file into the program, and others work with true dynamic link libraries, where intrinsic function numbers are used to determine the function in the library. For systems that do not support true dynamic linking, a slow replacement has been developed. This replacement involves taking the information necessary for the simulation, writing it to disk, calling the PDF simulator and running it on the information, and loading the revised information back into memory. The unfortunate problem with this method is that it can be anywhere from 30% to 3000% slower than the true dynamic linking method, depending upon the exact system architecture.

The code as it was developed initially supports file based PDF simulators, with the code for faster interfaces to be developed after the initial port. This pattern was followed with both the DOS and MS-Windows ports of CONSOL.

3.2 Support for external simulators

The creation of generic, system independent support for external simulators is not yet complete. At present, with the need to support MS-DOS and MS-Windows versions of CONSOL, generic single-tasking support for external sim-

ulators has not been completely designed or integrated into the overall system. The traditional UNIX support for external simulators is still in place, and the work is continuing to create a MATLAB and WATAND interface for the PC versions of CONSOL. The biggest limitation has been the single-tasking nature of MS-DOS and MS-Windows. However, limited success has been achieved, and full external simulator support should be ready in the near future.

3.3 User interfaces

As the user interface is the most important part of any system, this portion of the system has the most testing of any element of the system. The only way to test the rest of the system is by using the user interface, and when additional functionality was added to the engine, additional elements were added to the user interface to support the new features. The user interface is still developing and evolving as the beta testers for the "Portable CONSOL" project provide feedback.

4 Future development

4.1 Short term

The short term development of the system will include additional simulator support (especially in the area of external simulators), additional refinements to both the text-based and the graphical user interfaces, as well as incorporating additional features from the various UNIX derived versions (such as gradients, better control of functional constraints, etc).

4.2 Long term

There are still some machine dependent sections of the code that can be further modularized. Also to be explored is the concept of a graphical interface for tradeoff exploration. Such an interface would include be a visual (graphical) way to explore the various tradeoffs which can occur during the optimization process. For example, this interface would include methods for adjusting functional objective and constraint curves during optimization, methods for visually adjusting (relaxing and strengthening) the good and bad values of various objectives and constraints, etc.