Dr. Thevenin: An Intelligent Tutoring System for Electrical Circuits

by

E. Subrahmanian, S. Talukdar, W. Mullen

EDRC-05-12-87

Dr. Thevenin: An Intelligent Tutoring System for Electrical Circuits

Eswaran Subrahmanian

Sarosh Talukdar

William Mullen

Carnegie Mellon University

Pittsburgh,PA,15213

December 1986

To Appear in Knowledge Based Systems in Engineering, D. Shram, ed.

This work was supported in part by the Engineering Design Research Center and the Fund for Improvement of Post Secondary Education

ABSTRACT

Science and Engineering course contain one or two topics that are "bottlenecks" in that they are brief, very difficult and important. Students often fail to master these topics and encounter difficulty in future performance and learning. Thevenin Equivalents in electrical circuits is one such topic. In this paper, we have presented an architecture of a tutoring system based on artificial intelligence techniques of pattern matching and, divide and conquer for teaching Thevenin equivalents. The architecture consists of several modules that communicate through a common data base or blackboard. The modules are problem solver, problem generator, instructor, scheduler-input recognizer, and Thevenin expert. The system is implemented in OPS5, Lisp and C. The system uses the Andrew window manager system for providing graphical capabilities to represent and manipulate the circuit. The paper also presents an illustration of interaction between the student and the current implementation of the system.

Table of Contents

List of Figures

List of Tables

Dr. Thevenin: An Intelligent Tutoring System for Electrical Circuits

Eswaran Subrahmanian, Sarosh Talukdar, William Mullen
Engineering Design Research Center
Carnegie Mellon University

1 Introduction

Science and engineering courses commonly contain one or two topics that are "bottlenecks" in that they are brief, very difficult (students often fail to master them) and very important (the failure to master them limits the students' future learning and performance). Examples include:

- Thevenin equivalents in basic electric networks;
- Relation between force and acceleration in basic mechanics;
- Solution equilibria in chemistry;
- Relations among distributions and their characterizing parameters in statistics;
- Detailed mechanisms of alkaline reactions in organic chemistry;
- Maximization problems in calculus.

Any instructor in these areas can verify that year after year many students never master these topics. In two cases (forces and statistical reasoning) this lack of understanding has been well documented even in good students who have completed relevant courses [7,20]. These students were almost certainly hampered in any further work they undertook in these areas.

There are relatively few bottlenecks; seldom more than one or two per basic course. This, together with their other properties - brevity, difficulty and long range impact - make them prime candidates for concentrated research and development.

One way to handle a bottleneck topic is to make large numbers of good human tutors available when it is being covered. The shortage of good engineering and science faculty makes this approach unattractive. Recently, another approach has become feasible. It is to use artificial intelligence (AI) techniques to develop computer-based tutors (for an overview of computer-aided instruction and application of artificial intelligence to this field refer to the Handbook of Artificial Intelligence by Avron Barr and Edward Feigenbaum [1]), In this paper we describe the design and implementation of a Thevenin equivalent tutor using the second approach.

We have selected the Thevenin equivalent not only for its importance to Electrical Engineering but also because the process of forming a Thevenin equivalent involves a set of skills that are common to a wide variety of important technical and scientific problems. Specifically, students must learn to use topological rearrangements of networks, invoke conservation laws, formulate and solve simultaneous equations and deal with situations in which there are a very large number of different ways to arrive at a solution. By developing a computerized tutor to handle these aspects we will have obtained enough understanding and many of the basic parts from which to assemble other useful tutors.

In Section 1.1, we describe the purpose of tutors. Section 1.2 briefly review intelligent computer aided instruction efforts. Section 2 of this paper describes the Thevenin equivalent concept including an example. Section 3 describes the architecture of the tutor in terms of its components and their functions. Section 4 describes different tutoring strategies with which we plan to experiment. The current system implementation is described in Section 6 and contains several sample dialogues between a student and the system.

1.1 Tutors

A tutor is an instructor who works with one student at a time. The benefits of tutoring are:

- The instruction can be tailored to, and focused on, the student's needs. This is important because different students have different backgrounds, capabilities, and needs. They have different impediments to learning, need different mixes of remedial education and learn at different rates.
- There is instant feedback from the student. When the student does not understand something he can point it out. He can ask for better explanations or elaborations. The tutor may notice that the student is having trouble with some aspect of the problem and he needs more problems to practice. Instant feedback is especially important here, since much of engineering and science learning is done through problem solving, but to be effective, the problems must be neither too easy nor too difficult and must focus on issues with which the student needs experience. In a comparison between human tutors and traditional classroom instruction, Anderson and colleagues estimate that human tutors are well over 100% more effective than classroom instruction [2].

The main disadvantage of the tutoring approach to instruction has been the dearth of human tutors, even for educational bottlenecks. This is the major motivation for us to undertake this project.

1.2 Intelligent Tutoring Systems

The main difference between the earlier computer aided instruction approaches and the AI based approaches is that the tutoring system is capable of solving the problem as the student would solve the problem. In this approach the system is capable of identfying the student misconceptions based student solution as well as systems solutions to the given problem. Within this approach, Farrell and Anderson's LISP tutor uses a catalogue of student errors along with the model of an ideal student in the tutoring process[24]. Several tutors have been built in the domain of tutoring programming languages such as PROUST and MENO II[23,24]. Research in programming language tutors have concentrated on the cognitive aspects of programming as the basis for building tutors.

There are a number of artificial intelligence based computer-aided instruction systems in fields other than programming languages such as: SOPHIE [18], GUIDON [12], West [21], and MACAVITY [26]. SOPHIE performs question answering, hypothesis verification and theory formation activities in the domain of electronic troubleshooting. GUIDON, which is based on MYCIN, imparts the medical knowledge of MYCIN to medical students. WEST teaches students arithmetic operation in the context of the game of "How the West was Won." These systems focus more on the Artificial Intelligence aspect of the research than on cognitive and student oriented issues, and they provide a good background for work in this area.

2 Thevenin Equivalents

2.1 Rationale

Thevenin equivalents were selected as the first bottleneck to address for the following reasons:

- Importance: All electrical engineering undergraduates ought to master the skills involved in calculating Thevenin Equivalents early in their programs. But many do not and are hampered by their ability to appreciate succeeding material.
- Feasibility: We know from experience that even poor students can be taught Thevenin equivalents with just a few hours of tutoring. We feel that much, if not all, of the tutoring knowledge can be represented by rules. Also, the interactions between the students and the tutor can be conducted in a terse language - that of electrical circuits. Thus, the difficulties associated with building a natural language interface can be avoided.
- Generality: A wide variety of circuit and network problems can be handled by techniques that are either the same, or close to, those required for assembling Thevenin equivalents.

In summary, there are topics that are more important, topics that are easier to deal with, and topics that are more general than Thevenin equivalents, but a few that strike as good a compromise among these conflicting attributes.

2.2 The Problem

The Thevenin equivalent covers techniques for replacing a complicated electrical network containing many elements with a simple equivalent containing only two elements: a voltage source in series with a resistor[19]. As a typical example, a student is-presented with a circuit similar to the circuit in Figure 1 and asked to find out the Thevenin equivalent for that circuit considering the two nodes n1 and n2 as the two Thevenin terminals, i.e. the equivalent circuit shown in Figure 2. The behavior of the two circuits as far as the two terminals n1 and n2 are concerned are the same. The student's task is to find the value of the voltage source and resistance of the resistor shown in Figure 2. The value of the voltage source in Figure 2 is equal to the voltage measured across the two terminals n1 and n2 of the circuit in Figure 1. It is referred to as "open circuit voltage" and is represented as "voc." The resistance value for the resistor in Figure 2 is equal to the resistance seen across the two terminals n1 and n2 of the circuit in Figure 1 when all the independent voltage and current sources are eliminated (independent voltage sources are shorted, and independent current sources are opened) from the circuit of Figure 1. It is referred to as "Thevenin resistance" and is represented as "rth."

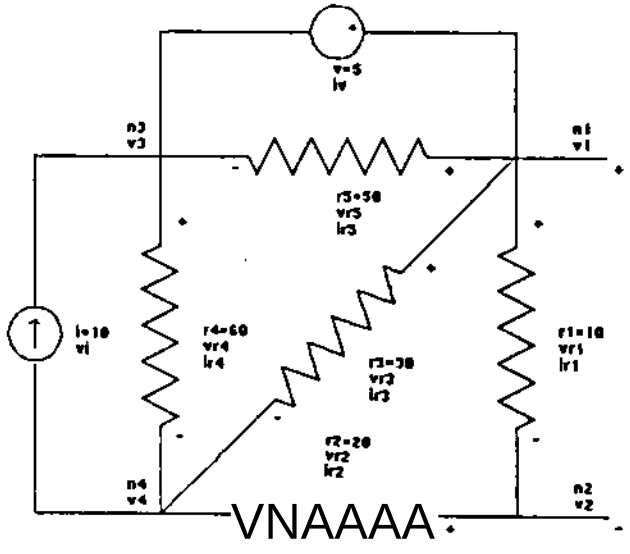


Figure 1: An example circuit

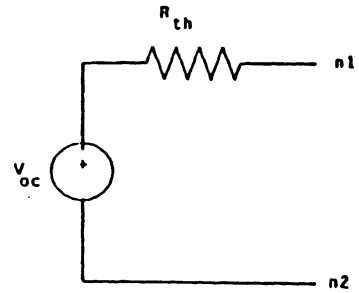


Figure 2: Thevenin equivalent for circuit of Figure 1-1

There are several ways to calculate voc and rth. To find voc, one can use either of the following two methods:

- Write node equations for the circuit as it is and calculate voc (voc=voltage at node n1 minus voltage at node n2).
- Short the two terminals n1 and n2; find the current in the wire connecting these two nodes (this current is referred to as "short circuit current" and is represented as "isc"); and then use the relation voc=rth*isc, provided that rth is also calculated.

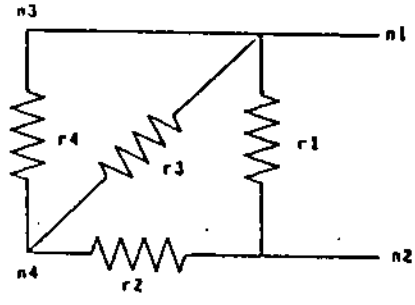
To find rth, one can use one of the following methods:

- Eliminate all the independent voltage and current sources (short independent voltage sources and open independent current sources) and use network transformation to transform the circuit to a new circuit which has one resistor connected between the two nodes n1 and n2. The value of this resistor is the same as rth.
- Short the two Thevenin nodes n1 and n2; find isc and then use the relation rth=voc/isc to calculate rth, provided that voc is known and isc is not zero.
- Eliminate the independent voltage and current sources; connect an arbitrary known current/voltage source across the two terminal nodes n1 and n2; find the voltage/current for the added source, rth is equal to the ratio of the voltage to the current for the added source.

Based on the above explanation, finding the Thevenin equivalent for a circuit involves the following subtasks: choosing the method to be used, network transformation, writing node or branch equations, and finally solving simultaneous equations. Figure 3 shows the procedure for finding the Thevenin equivalent circuit for the circuit of Figure 1. The givens are the value of the five resistors (r1=10, r2=20, r3=30, r4=40, r5=50 ohms), the voltage of the independent voltage source (v=5 volts), and the current of the independent current source (i=10 amperes).

To Find voc:
set v2 = 0 (reference voltage)
voc = vl (open circuit voltage)
vl-v3 = v (branch voltage)
vl/rl+v4/r2 = 0 (node equation for node n2)
i+(v4-v3)/r4+(v4-vl)/r3+v4/r2 = 0 (node equation for node n4)
solving the above equations yields: voc =40.33

To find rth:
short vl
remove r5
open i
The new circuit is:



replace two parallel resistors r3 and r4 by r6 =(r3*r4)/r3+r4) = 20
replace two series resistors r7 = r6 + r2 = 40
rth = (r7*rl) / (r7+r2) = £.j)

Figure 3: Solution for voc and rth for the circuit in figure 1

3 Architecture of Dr. Thevenin

The tutor consists of the following components: Problem Generator, Problem Solver, Student modeler, Instructor, User Interface and Thevenin Expert. The architecture of the the tutor is presented in Figure 4. All the components of the tutor are written in OPS5 and LISP except for the User interface that is written in C. Each of the components of the tutor is described in the following paragraphs.

3.1 Problem generator

The task of the problem generator is to generate problems dynamically. The class of problems generated will correspond to a curriculum developed by expert human teachers and also based on students requests and student's present knowledge; gathered by other components of the tutor. A provision is also made through the User Interface to allow students to enter a network

problem that he would like **to solve.**

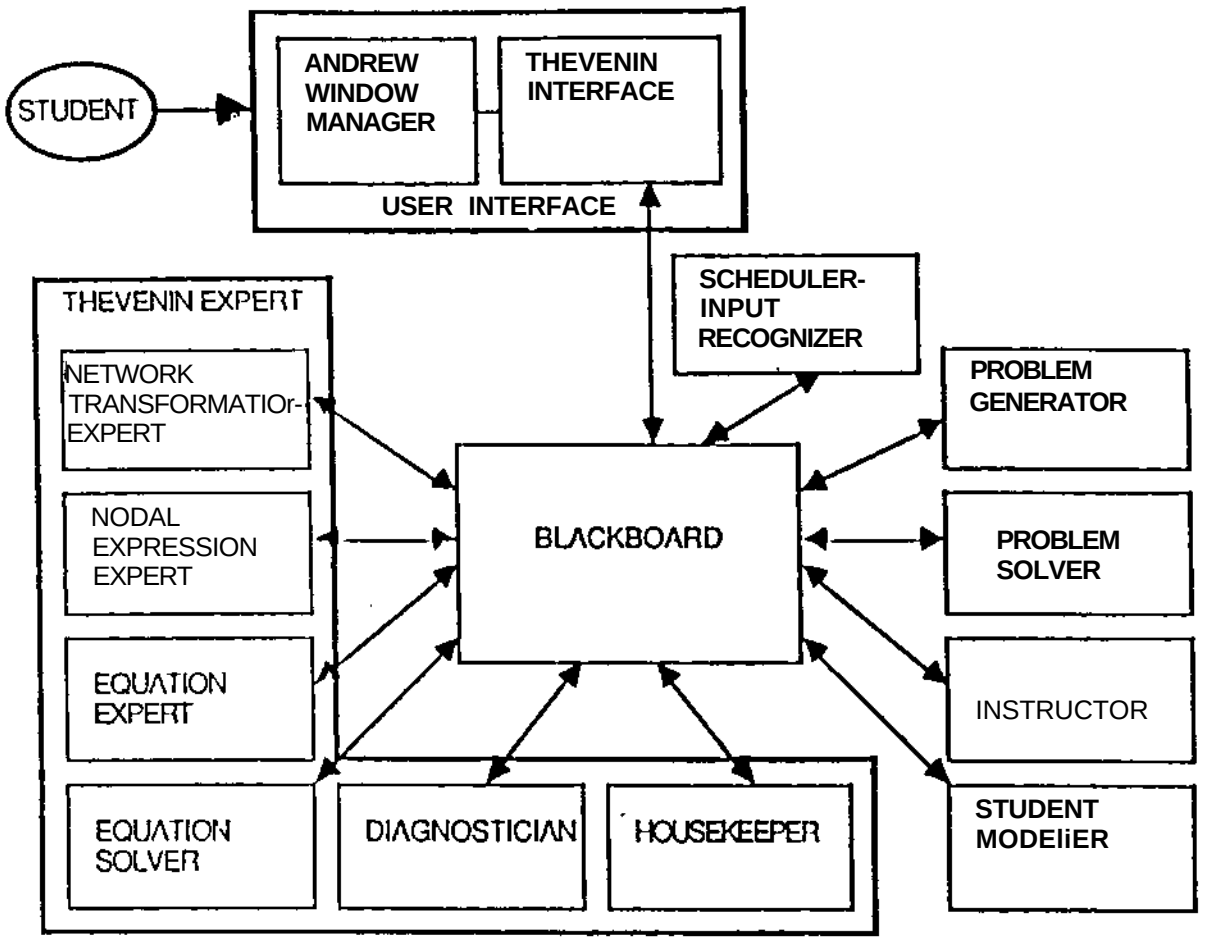


Figure 4: Architecture of Dr. Thevenin

3.2 Problem Solver

Teaching objectives can best be achieved in the problem solving context[2]. Since problems will be created or entered dynamically, the system should be able to solve them dynamically. in our context, it implies that the tutor is able to solve a given problem in the most efficient fashion that reflects the objectives of teaching all the different concepts of the problem solving task. The problem solver is used in two different modes. In the first mode, the problem solver would be used to provide illustration of the problem solving process on for a given problem. In the second mode, the task of the problem solver is to generate an ideal solution to a given problem. This solution will be used by the Student Modeler in identifying concepts which the student does not utilize in his solution. The problem solver also has the task of providing help to the student, if at any time he is not able to proceed, by suggesting a set of steps that he could take towards the

solution.

3.3 Student Modeler

The purpose of the student modeler is to model the student's understanding of concepts in the domain. The student modeler has a set of criteria on which the student is judged [13,14]. These criteria are derived carefully by looking at the concepts and knowledge that should be learned in finding the Thevenin equivalent of a circuit. Some of these criteria are based on the knowledge of:

1. **voc=rth*isc;**
2. **network transformations;**
3. **how to write a correct node equation;**
4. **the correct set up for finding voc, rth, isc;**
5. **branch voltage equations; and**
6. **Ohm's Law.**

The student model generated is used in exercising the student in concepts that he is not familiar with by generating problems that emphasize the use of those concepts. The student modeler works at two different levels. At the first level, the modeler utilizes information generated by the diagnostician, which identifies student errors in individual steps, to identify concepts that the student is weak in. At the second level the student modeler compares the solution generated by the problem solver with the student solution to identify concepts that the student is expected to use but fails to do so.

3.4 Instructor

The purpose of this module is to create explanations which are robust, concise and relevant to the most recent entry of the student. It also grades the solution generated by the student. This module directly interacts with the student and all other modules generate information that is used by the Instructor to interact with the student. This separation allows the imnplementation of two distinct modes of the Thevenin expert. The two modes, active and passive, correspond to immediate instructional mode and post-solution diagnosis mode. This issue is explored further in the section on tutoring styles.

The grader part of the Instructor will be operational only in the passive mode of the Thevenin expert. The grader will use relative importance of concepts learned and not understood as a

means to assign a numerical score to the student solution. For example, a student may solve the problem of determining Thevenin equivalents correctly without using any of the network transformations. This may be due to his lack of knowledge of network transformations that are important from the point of view of simplifying large circuits and reducing the number of simultaneous equations to be solved and hence will bbe graded accordingly.

3.5 Scheduler-Input Recognizer

The Scheduler is the control routine that schedules the different expert modules of the Thevenin system. The student input is also identified and categorized before invoking the appropriate expert to execute the input. The student input can be broadly classified into commands and expressions. If the input is a command, the appropriate expert is scheduled based on whether the command is a network transformation command or a high level system command. For example, in the case of restarting a problem or changing the current goal all the temporary data structures generated in the course of problem solving have to be discarded. This task is assigned to the housekeeping expert before the student can be allowed to enter any problem step. The scheduler is also responsible for activating the grader, problem solver, the student modeler and problem generator based on the mode of interaction and the type of student requests.

3.6 User Interface

The problem solving procedure involves network transformation that is accomplished graphically while the nodal and branch equations require keyboard input. Students should be able to see the most recent version of the network drawing, modify the circuit easily, see all the equations entered. In order to provide a user friendly interface, the Andrew Distributed Personal Computing environment developed by the Information Technology Center (a Joint Project of IBM and CMU) is used [25]. The Andrew system has a window manager, implemented on a variety of machines with a high resolution monitor, which provides a bit-mapped graphic display with a multiple and overlay window management facility.

The User Interface consists of a) Andrew window manager and b) Thevenin interface - a communication routine between the window manager and the Thevenin expert. The interface is connected to a data base of text skeletons that are used in generating the information to be

conveyed to the student. The pupose of the interface is also to update the graphical displey of the network based on the network transformation commands. The display presented to the student consists of a window with three distinct partitions: a) Equation window b) Interaction window and c) network window. The Equation window is used to present the set of correct nodal and branch equations. The Interaction window as the name suggests is where all the interaction between the student and the Instructor module takes place. The network window is used to show the current status of the network in the problem solving process. Examples of sessions is provided in a later section where the use of these windows will become apparent (figures 5 and 6).

3.7 Thevenin Expert

The Thevenin expert is the set of routines which participate in the interaction with the student in identifying the student input, analyzing the nodal equations, solving a set of equations, performing network transformations, and diagnosing student errors. Each of the modules of the Thevenin expert is explained in the following paragraphs.

3.7.1 The Equation Expert

The purpose of this expert is to check the equation input by the student to determine what type of relation it represents, and to determine whether it is correct. There are numerous types of equations that are allowable in the domain, among them are reference equations, Ohm's Law, branch voltage relations, and nodal equations. A combination of LISP functions and OPS5 rules are used for equation transformation and recognition. LISP functions are used to transform the equation using mathematical knowledge consisting of algebraic rewrite rules. OPS5 rules (condition-action rules) are used in the equation recognition process. Each rule represents one piece of knowledge; in this area it represents knowledge about one type of equation or mistake.

The Equation expert, first transforms the given student equation into a standard form. This standard form of infix notation, is a sum of terms, each with a sign, operator, and one or two operands. The standard form is generated by bringing all of the terms to one side of the equation, and applying the distributive law and other algebraic knowledge. The result of the transformation is stored as a set of working memory elements that is easy to check against standard forms of correct equations. For example, an Ohm's Law equation for resistor ri in Figure 1 would be : (v1 - v2) / r1 = ir1. This equation is broken down by the system as (+ vi / ri)

+ (- v2 / r1) + (- ir1) = 0. Then these elements match a rule specifying one of eight forms for Ohm's Law in a single step. There are several rules for Ohm's Law since the equation could also have been entered as vr1 / r1 = ir1 , ir1 * r1 = vr1 , v1 / r1 = ir1 (if v2 is ground) and so on. A branch voltage relation or reference equation is recognized in similar fashion. Use of the standard form vastly reduces the number of patterns which must be checked by the rules in OPS5. For example, the associative law is used implicitly to break down either (a + b) + c = 0 or a + (b + c) = 0 into the form (+ a) + (+ b) + (+ c) = 0; they are represented identically in the working memory.

A more complex case is that of replacement equations. When the student wishes to replace two series or parallel resistors with an equivalent one, he enters the equivalence relation for the new resistor. For two resistors in series, the equivalence relation would be of the form r1 + r2 = req, but for parallel the relation can be one of many, among them are: req = r1*r2/(r1+r2), 1 / req = 1 / r1 + 1 / r2, req = 1 / (1 / r1 + 1 / r2). These are recognized in a two-step process. First, if the terms match a pattern for series or parallel replacement, the scheduler invokes the network transformation expert to check the validity of the replacement. Other rules then check whether the two resistors exist, and whether they are in the proper orientation for replacement. If the replacement is valid, the graphical display of the circuit is updated to show the new configuration.

If the student equation is none of the above valid voltage, current, or replacement relations, a diagnostic procedure to determine what each term represents, and whether there are any errors in the equation is invoked. The units of each term are determined and classfied into voltage, current, resistance, or unknown. At this point any mistakes such as wrong sign, or wrong branch voltage are detected by the diagnostician and brought to the user's attention. If the equation is a sum of currents, it is assumed to be Kirchhoff's Current Law (KCL) for some node, and the user is queried as to which node the equation corresponds to. This invokes a separate expert for nodal equations. The student can speed up this process by issuing a command "node n2" first, which starts the nodal expert directly after the equation is entered circumventing the equation recognition part of the Equation expert.

3.7.2 **The Diagnostician**

The tutor's diagnostician works in all areas of instruction. It is invoked when the student tries to perform an illegal network transformation, such as eliminating independent sources while finding the open circuit voltage. It keeps track of the number of equations that the student has entered and the number of unknowns in them, so that if the student tries to solve three equations for five unknowns, he or she is asked to write more equations. The diagnostician is especially useful to identify errors detected by the equation and nodal equation experts. Here the user is informed of such mistakes as trying to add quantities of two different units, missing terms, inconsistent sign, etc. If the student enters an equation that is completely wrong, the diagnosis will fail and the user will be told that his equation is wrong. For nodal equations the error checking is very precise. If the student makes several mistakes in a single equation, the diagnostician lists all of his errors. If a term in the nodal equation is wrong, the system points out the correct version of the term.

3.7.3 Nodal Equation Expert

The nodal equation expert deals with an important type of equation which is basic to circuit analysis: Kirchhoff's Current Law, which states that the sum of currents into (or out of) a node add to zero. A separate expert is useful for this purpose because the node equation can be written in many forms, but the diagnosis is similar for any node in any form. Consider the circuit in Figure 1. A valid node equation for node n4 is : (v4-v3)/r4+(v4-v1)/r3+v4/r2+i=0 , with all of the currents out of the node and node n2 the reference node. However, it is also valid to reverse the sign of each term, representing the sum of currents into the node. Substituting - ir4 for (v4-v3)/r4 in the equation is also acceptable due to Ohm's law . Note that the current through resistor r2 is v4/r2 since v2 is zero (reference), but writing (v4-v2)/r2 is acceptable.

The nodal expert uses the standard form generated by the equation expert in determining the correctness of the given nodal equation. If the student fails to identify the node for which the equation corresponds, the system prompts the user to identify the same. The system using this information generates a version of the nodal equation for the specified node in a standard form. The terms in the student equation and the system's equation are matched term by term. This leads us to four different situations: 1) all the terms in both equations are matched; 2) there are additional terms in the student equation; 3) there are additional terms in the system equation; and 4) there are unmatched terms in both student and system equations. The first case is trivial in that

student and system equation are identical. The second case has two subcases in that the student equation contains additional terms that reduce to zero on value substitution and hence correct or it is an unidentifiable term. The third case corresponds to a situation where the student has left out terms corresponding to some components attached to the node. The fourth case is in non-trivial in that the system now attempts to substitute equivalent terms for the additional terms using Ohm's Law or circuit configuration as in the use of a current term based on series orientation of the components, to match the excess terms in the student case. At the end of the substitution and matching process (a hypothesize and test cycle), the state of the of the nodal expert is in any one of the situation listed. However, the fourth case is different when all the possible substitutions have been tested. At this point it can be inferred that the student has terms that are in error and may also be missing terms. In the cases of incomplete matches, the nodal equation expert works with the diagnostician to determine whether terms are extra, missing, or wrong and point to the student the corrections required to render the equation valid. Specifically, if the student wrote the wrong current through a resistor, a message will be printed such as: "Wrong current for component r1. It should be: (v1 - v2)/r1."

3.7.4 **Housekeeping** Expert

Various experts such as the equation and nodal equation expert, diagnostician, in order to keep track of the current state of problem solving or the recognition process create many temporary working memory elements. If an equation that was entered is found to be incorrect, the working memory elements generated in processing that equation must be removed. If a student has transformed the circuit while solving for a Thevenin variable and wishes now to solve for another Thevenin variable, the circuit must be returned to its initial state. In these instances, the control rules schedule the housekeeping expert. Depending on the type of cleanup needed, it can remove components from the circuit that have been added, eliminate the equations that have been entered, or simply remove unnecessary memory elements.

3.7.5 Network Transformation Expert

This module of the tutor represents knowledge about the transformations which are permitted depending on the Thevenin variable the student wishes to solve for. The student can issue the following transformation commands using a mouse or the keyboard:

```
short c      : short a component named c
open c       : open a component named c
```

```
                (usually an independent current sources)
remove c    : remove a component named c
                (same as open but for any component)
original    : restore the original circuit and start over
vtest       : add a test voltage source to the Thevenin terminals
                (for dependent sources)
itest       : add a test current source to the Thevenin terminals
                (for dependent sources)
```

Replacement transformations are entered by typing the equivalence relation directly, such as:

```
req = rl + r2              series replacement

req = rl*r2/(rl+r2)        parallel replacement
```

When any of the above commands or equations are entered, the transformation expert is scheduled to determine whether it is a valid step. This involves making sure that the component(s) exist, that the goal is correct and they are in the correct orientation. For example, when calculating short circuit current, a resistor can be shorted only if it is across the Thevenin terminals . Independent sources can only be eliminated only when solving for rth. If the transformation is found to be correct, the circuit is updated on screen and the user is informed. Otherwise, the diagnostician is invoked informing the student of his mistakes.

3.7.6 Equation Solver

The purpose of this module is to ensure that the subset of equations chosen by the student to solve for the unknown quantities is an independent set. The module does this by identifying the number of unknowns and the number of equations submitted for solving. If satisfied, The set of equations are handed over the symbolic math package Macsyma or to a Gauss elimnation procedure[10]. Macsyma solves the the set of equations if they are an independent set, else returns an error. If an error is returned the student is informed of the error by a rule in this module of the error, else the values of the unknown variables are displayed.

4 Tutoring Style

A tutor is a an agent that observes student's behavior, finds student's mistakes, misunderstandings, and lack of knowledge, and conveys this information at the appropriate time using a relevant explanation to the student. Deciding upon the appropriate information, time and explanation involves several issues: what is the knowledge one wants to teach the student; what does the student know presently, and what has been discussed before (student modelling); what is the best time that the information is relevant for discussion and student is ready for acquiring it;

what is the best sentence to convey information (text generation [11]). There are a number of computer tutors that have addressed these issues to varying degrees [12,21]. At this stage we have implemented two styles that are based on the following assumption.

There are two distinct stages in learning a new subject: learning the concept at the beginning and learning the subject by practice after the concept is known. The interaction between the student and the human teacher, for the first stage of learning, involves the student declaring every step that he takes and the teacher responding immediately and acknowledging the validity or invalidity of the student's response. The two major reasons for this type of interaction are: first, the student develops a strategy and methodology for the problem solving process; second, the teacher prevents the student from forming bad habits or misunderstanding the subject.

The interaction between the student and human teacher, for the second stage of learning involves assigning and correcting homeworks. Students are given homework assignments and asked to solve and return it. Students might take the wrong path, or write wrong equations and recover from some of them and correct them in the final copy handed in. During this process there is no immediate comment from the teacher. In this stage students learn mainly from practice and from their own mistakes.

With respect to the system's explanation to the student, there are two different modes: immediate response (active) mode, and delayed response (passive) mode. In the active mode, the system checks the student's input and acknowledges its validity and correctness before the student is allowed the next step. In the passive mode, the system does not respond to student's input immediately, it lets him do what he thinks is right. When the student gets to the point where no more mistakes are tolerable, or he declares that he is through with the problem, the tutor intervenes and points out the student's mistakes. In this mode of tutoring the student has a chance of correcting his mistakes before the tutor intervenes. We will be experimenting with the two modes of tutoring to identify their deficiencies at different stages of learning to augment/modify the tutor based on the principles suggested by Anderson[2], Guidon[i2], and WEST[21] projects.

5 System Knowledge

The system has two distinct types of knowledge: mathematical knowledge and electrical engineering knowledge.

The tutor's mathematical knowledge includes knowledge required to rewrite algebraic expressions. They are :

- transitive law [a+b = b+a, a*b = b*a].
- associative law [a+(b+c) = (a+b)+c].
- distributive law [a*(b+c) = a*b + a*c].
- equivalence [a+0 = a, a*1 = a, a*0 =0]

These laws are represented in a combination of OPS5 rules as well as LISP routines. As the purpose of the tutor is not one of teaching math principles one can safely compile this knowledge in a procedural or functional language.

The system's electrical engineering knowledge, being the domain of the tutor, requires that it be coded in a non-compiled form. The tutor has the following knowledge of electrical circuits:

1. Calculations: The tutor knows the method to calculate open circuit voltage, short circuit current, and Thevenin resistance and several methods of computing the the three attributes of the Thevenin equivalent (voc, isc, rth).
2. Components: The present tutor knows of resistors, independent and dependent voltage and current sources. The final version is expected to have capacitors, inductors, etc.
3. Measurements: The tutor uses the measurement units of the attributes, current, resistance, conductance and voltage, in identifying student's misconceptions on the relationships. As new components are added new units of measurement and their branch relations will also be added to the system.
4. Network Transformations: The tutor has the following network transformations currently available:
    - replacing two series or parallel resistors with an equivalent.
    - replacing two series voltage sources with an equivalent.
    - removing shorted current source or resistors.
    - removing a voltage source in parallel with the current source, aggregating parallel and equal voltage sources.
    - addition of test voltage or current source.
5. Expressions: The tutor recognizes the following types of expressions (examples based on the circuit in Figure 1-1):
    a. Node equations (Kirchhoff's current law). Example: node equation for node n4 can be written as: i+(v4-v3)/54+(v4-v1)/r3+(v4-v2)/r2=0.
    b. Branch equations. Example: Branch current for resistor r1 (ir1) can be written as: ir1=(v1-v2)/r1.

c. Ohm's Law. Ohm's law represents the relationship between component current and voltage and can be used to compute them. Example: voltage for resistor r1 (vr1) can be written as vr1 = ir1*r1 and the current for resistor r1 is ir1 = vr1/r1.

d. Open circuit resistor voltage is 0.

e. Transformation equations. To replace two parallel resistors r3 and r4 with an equivalent resistor req one can write: 1/req = 1/r4+1/r3.

f. Reference Voltage equation. Choosing v2 as the reference voltage is accomplished by writing v2 = 0.

The system currently is capable of using this knowledge only in identifying whether the student input is a valid one given the current status of the circuit and to perform any of the topological transformations if appropriate. The system is subsequently expected to use this knowledge in generating a solution for any given problem. This knowledge is required for guiding a student through an example or to modify his problem solving sequence in order make use of all the concepts that are available to him during the problem solving process.

6 Illustration of Implementation of System Knowledge

In this section we illustrate by examples the implementation of system knowledge as OPS5 rules. For this purpose we have chosen two examples: 1) a network transformation example, and 2) Ohm's Law. The first example has 2 rules that are used in identifying the whether a given network transformation equation for the replacement of two resistors in series is correct. These rules assume that the student equation, written in the form of req = r1 + r2, for replacing two resistors in series, has been transformed into a set of OPS5 working memory elements consisting of the unary terms in the equation. Given these terms are available, the first rule (figure 5) is used to identifythe student equation for replacement of two components that are in series. This rule also sets up the context for checking the correctness of the replacement equation for the two resistors. The second rule (figure 6) checks to see whether the two resistances chosen in the student network transformation equation are in series in the given network. If so, a set of working memory elements corresponding to the replacement component added to the network is generated while the replaced resistors are discarded. A call to the graphics routine to make the transformation on the screen is issued by another rule. The two rules in figures 5 and 6 illustrate the case where a correct series transformation equation has been written. There are other rules that identify the incorrect application of the series replacement equation. As can be seen from the rule in figure 5, only conditions for the case of two elements in series are specified. If there are

more elements in series, the system using these rules would require that the transformation be done in pairs.

```
;  IF exactly two unary terms with the same sign,
;  and one unary term of opposite sign are present,
;  and the two similar terms refer to components
;       in the circuit,
;  THEN  assume the equation is a series replacement
;  and change context to objective replace in order
;       to check validity of the replacement.

(p equation-might-be-series-replacement
(context ^objective equation ^report <el>)
(eterm ^sign <sl> ^id <idl> ^opl <rl> ^operator nil)
(eteim ^sign <sl> ^id (<id2> <> <idl>} ^opl <r2> ^operator nil)
(eterm ^sign <> <sl> ^id <id3> ^opl <req> ^operator nil)
(component ^name <rl>)
(component ^name <r2>)
-(eterm ^id {<> <idl> <> <id2> <> <id3>})
-(student-error)
—>
(make context ^objective replace ^report. <el>)
(make replace-command ^type series ^compl <rl> ^comp2 <r2>
     ^compeq <req>)
(remove 12 3 4))
```

Figure 5: Rule for Recognition of Equation **for** Replacement of Two Components in Series

The second example (figure 7) illustrates the incorporation of Ohm's law for checking the correctness of an ohm's law equation written by the student. One version of the Ohms law equation is recognizable by the fact that the equations consists of three terms; two of which are of the form (voltage/ resistance) but with opposite signs and the third one is a current term. Here, if a given student equation is found to have the above structure, the rule for Ohm's law becomes: Given an equation of the form v1/r1 -v2/r1 -ir1 = 0, if the voltage variables correspond to the voltage variables of the two nodes to which the component r1 is connected and if the current term corresponds to the current term associated with the component r1 then the equation is a correct. The rule presnted is for one of several forms of the Ohm's law equation that can be wriiten for a component. The form of the Ohms law equation for a component depends on the choice of variables associated with the component and the nodes to which the component is connected. The rule illustrated uses the nodal voltages and the component current, while other forms may use component voltage and component current. In the case of a grounded node connected to a component, the voltage associated with the grounded node being zero allows one to write the equation without the term involving the voltage of the grounded node. There are rules in the system to account for the different forms in which Ohm's law can be written.

```
; IF the replace-command is series,
;     with two different resistors being replaced by a new resistor,
;   and the resistors have one common node with only 2 branches,
;   and the node is not a Thevenin node,
; THEN  replace those two resistors with a series equivalent named <req>
;   with value equal to the sum of the two resistor values.
;   A Lisp function called createsymbol is used to make current
;   and voltage names for the new resistor.  An explanation is
;   given to the user that the replacement is correct, and
;   which triggers the graphics function updating the circuit on screen.

(p series-replacement-is-valid
(context Aobjective replace Areport <el>)
(replace-command Atype series Acompl <rl> Acomp2 {<r2> <> <rl>}
    Acompeq {<req> <> rth) Astatus nil)
(component Aname <rl> Atype resistor ^status active ^protected nil
    Avalue <vl>)
(component Aname <r2> Atype resistor Astatus active Aprotected nil
    Avalue <v2>)
(node Aname <nl> Ano-of-branches 2 Aprotected nil)
(comp-node Acomp <rl> Anode <nl>)
(comp-node Acomp <r2> Anode <nl>)
(comp-node Acomp <rl> Anode {<n2> <> <nl>})
(comp-node Acomp <r2> Anode {<n3> <> <nl> <> <n2>})
(orig-exp Aid <el>)
(node Aname <nl> Ano-of-branches 2 Aprotected nil)
-(comp-node Acomp $thevenin Anode <nl>)
-(component Aname <req>)
-(component Avoltage <req>)
-(component Acurrent <req>)
—>
(make context Aobjective interact)
(modify 3 Astatus replaced)
(modify 4 Astatus replaced)
(modify 5 Al deleted-node)
(remove 1 2 10)
(call createsyxnbol i <req>)
(cbind <cc>)
(bind <c> (substr <cc> 1 1))
(remove <cc>)
(call createsymbol v <req>)
(cbind <w>)
(bind <v> (substr <w> 11))
(remove <w>)
(bind <rval> (compute <vl> + <v2>))
(make comp-node Acomp <req> Anode <n2> Asign +)
(make comp-node Acomp <req> Anode <n3> Asign -)
(make component Atype resistor Aname <req> Resistance <req>
   Astatus active Acurrent <c> Avoltage <v> Aadded yes Avalue <rval»
(make explanation Amsg-class correct Aurgency low Amsg-no 14
      Acontent Series <rl> <r2> <req>))
```

Figure 6: A Rule for Checking the Validity of Series Replacement of Resistors

```
;  Rule for OHM'S LAW equation of the form :  (vl - v2) / rl = irl
;  IF there are only three terms in the equation,
;     and two of them are node voltage divided by a resistance,
;     with opposite sign,
;     and a third term is unary
;     and represents the current of that resistor with correct
;        sign convention,
;  THEN  accept that equation as a correct Ohm's Law relation,
;     and inform the user.

(p equation-is-ohms-law
(context ^objective equation ^report <el>)
(eterm ^sign <sl> ^id <idl> ^opl <vl> ^op2 <rl> ^operator /)
(eterm ^sign <> <sl> ^id <id2> ^opl <v2> ^op2 <rl> ^operator /)
(eterm ^sign <> <sl> ^id <id3> ^opl <irl> ^operator nil)
(component ^naxne <rl> ^current <irl> ^resistance <rl> ^voltage <vrl>)
(coxnp-node ^sign + ^coxnp <rl> ^node <nl>)
(node ^nazne <nl> ^voltage <vl>)
(comp-node ^sign - ^comp <rl> ^node <n2>)
(node ^name <n2> ^voltage <v2>)
-(eterm ^id {<> <idl> <> <id2> <> <id3>})
-(student-error)
-(strategy ^type elimination)
-->
(make context *objective interact)
(call subst-knowns-in-eqn <el>)
(remove 12 3 4)
(make explanation ^urgency low ""xnsg-no 3 ^content <el> <rl>))
```

Figure 7: Rule for Recognizing a Version of Ohm's Law

7 Current Implementation and Sample Runs

The initial implementation of the system was done on a Vax-11/780 using OPS5 and LISP. Due to the lack of graphical facilities, the current implementation was carried out on the Sun workstation with the Andrew window manager [22]. Moving to this environment was motivated by both the existence of graphical facilities as well as to make the system available to large number of students through a campus wide network of personal computers. The initial version of the tutor consisted of about 700 OPS5 rules where the typical mean size of the conflict set was 90 rules. The current version has about 258 rules and the mean size of the conflict set is 15 rules. Table 1 shows the breakdown of rules by components of the system. The current prototype runs at three times the speed of the initial prototype system. The current version includes dependent current and voltage sources that was not present in the initial version.

The components of the system that have not been yet implemented are a) problem generator, b)problem solver, and c) grader. There is a simple version of the student modeler that only deals

Expert Module	No of rules
scheduler/recognizer	29
network transformation	20
equation expert	45
nodal expression	50
diagnostician	60
student modeler	20
housekeeping	20
equation solver	14
Total	258

Table 1: Breakdown of rules by Expert modules

with misconceptions about the concepts used by the student. In the next stage of development, the student modeler will be used to identify and correct the problem solving strategy of the student to incorporate concepts that the student fails to use in his solution.

In Figures 8 and 9 we present interactions between the student and the system for solving voc and isc for the circuit presented in figure 1 respectively.

The first example illustrates the behavior of the tutor when a student does not commit any mistakes in solving for the Thevenin voltage voc for the given circuit. In figure 8, the three windows, the interaction window (the window with the Student> prompt), the equation window and the circuit window can be seen. The equation window contains all the correct equations written by the student in solving the problem. The graphical window displays the current status of the circuit. As can be seen in figure 7the network transformation commands are available to the user in the form of a menu that can be chosen with a mouse. The system informs the student of the correctness of each of his steps.

In the example in figure 9, an attempt is made to show the interaction of the system to point out errors to the student. The commands at prompt 2 and 4 correspond to network transformation commands that are invalid in the pursuit of the current goal to find the Thevenin current isc. Prompt 3, corresponding to component replacement equation arises due to the misunderstanding of behavior between series and parallel resistances. The errors in the nodal equation entered are pointed out to the student term by term and in the case of nodal equation for n4, the system identifies the use of incorrect current term v4/r5 for the component r5 by displaying the correct term (v4-v6)/r5.

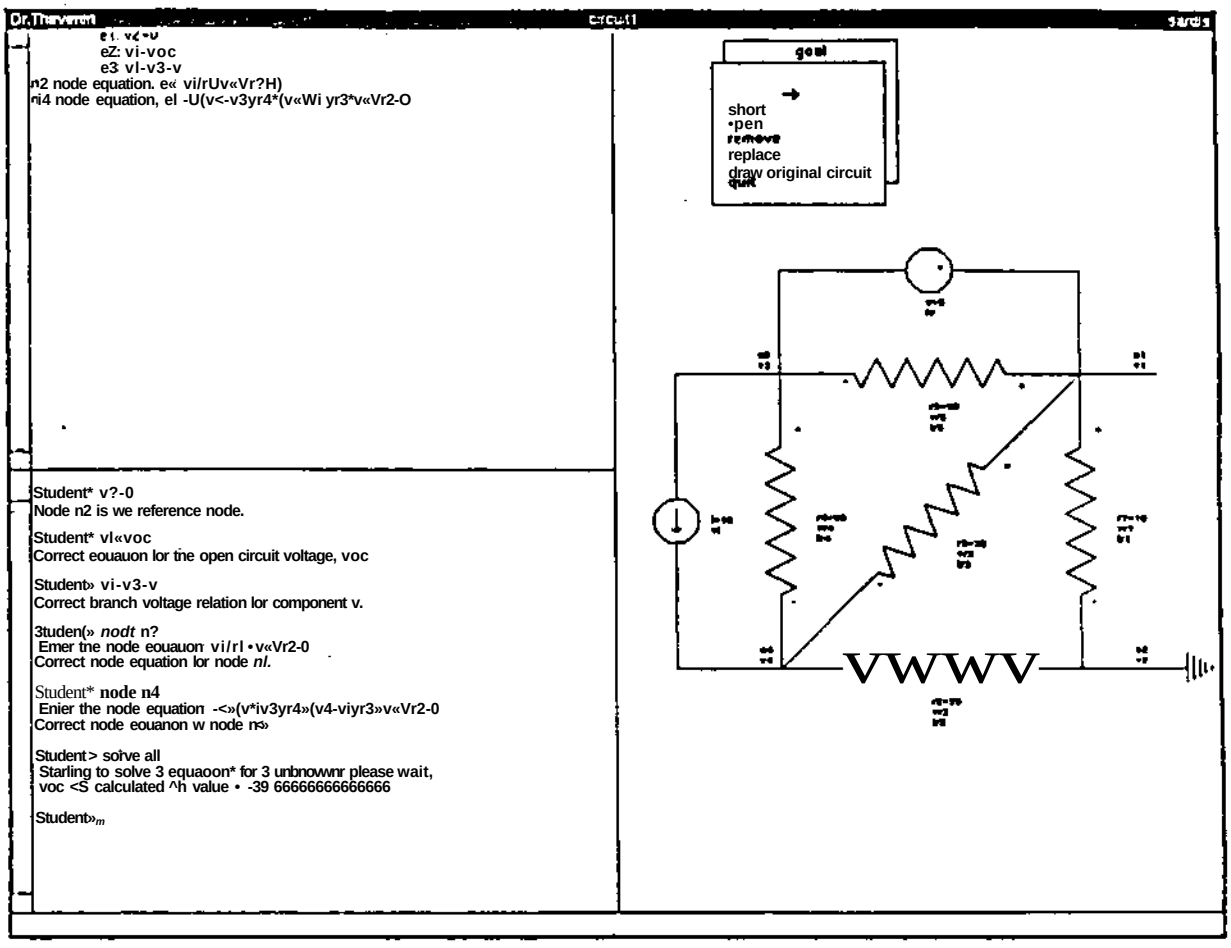


Figure 8: Calculation of voc without any errors

8 Summary

In this paper we have presented an architecture and implementation of an intelligent tutoring system for teaching Thevenin equivalents, a bottleneck in engineering education. The tutor architecture, based on the blackboard architecture, is robust in that many components of the present system can be used in other subjects in Electrical Engineering or even in disciplines other than electrical engineering. For example, circuit analysis can be used for any other electrical engineering subject, the mathematical part, the framework for tutoring style, problem generation and student modelling can be used in computer based tutors in a variety of domains.

We tested the first version of the tutor without the graphical interface and with graphical interface and found substantial interest in using the system provided the speed of the system could be reduced. The current version with an average response time of about 15 seconds will

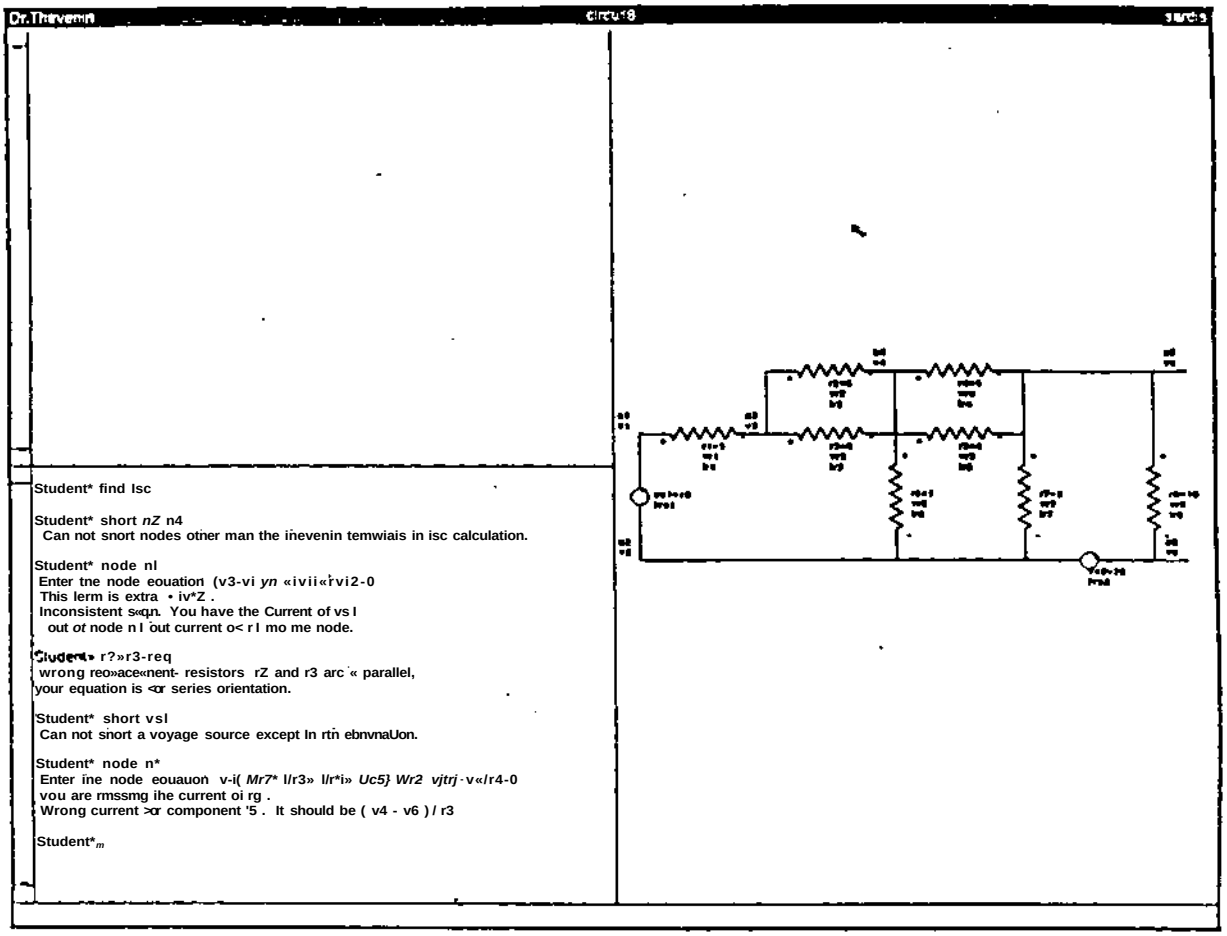


Figure 9: Calculation of isc with errors

be tested for acceptability. We do not intend to replace human tutors, but are confident that with the artificial intelligence techniques utilized in the implementation of the system we can handle most of the student needs - routine misunderstandings, the common place errors and most of the explanations they need. The use of a graphical interface offers the student a scratch pad to attempt to solve problems with relative ease while providing continuous monitoring of the student learning.

References

1. Barr, A., and E.A. Feigenbaum (Eds.) *The Handbook of Artificial Intelligence (vol 1,2,3).* Los Altos, CA: Kaufmann, 1981, 1982.

2. Anderson, J.R., C.F. Boyle, R. Farrell, and B J. Reiser. Cognitive Principles in the Design of Computer Tutors. *Internal paper, Advanced Computer Tutoring Projects Carnegie Mellon University,* 1984.

3. Davis, R., and B. Buchanan. Production Rules as a Representation for a Knowledge-Based Consultation Program. *Artificial Intelligence 8, pp. 15-45,* 1977.

4. Buchanan, B.G., and R.A. Feigenbaum. Dendral and Meta-Dendral; their Applications Dimension. *Artificial Intelligence 11, pp. 5-24*, 1978.

5. Forgy, C.L. *OPS5 User's Manual*, Department of Computer Science, Carnegie Mellon University, July, 1981.

6. Foderaro, J.K. *The FRANZ LISP Manual* University of California at Berkeley, 1981.

7. Green, B., M. McCloskey and A. Caramazza. Curvilinear Motion in the Absence of External Forces: Naive Beliefs about the Motion of Objects. *Science, 210(5), pp. 1139-1141*, December 1980.

8. Erman, L.D., R. Hayes-Roth, V.R. Lesser, and D.R. Reddy. The Hearsay-II: Speech Understanding System: Integrating Knowledge to Resolve Uncertainty. *Computing surveys, Association for Computing Machinary (ACM), Vol. 12, No. 2, pp. 218-253.* February 1980.

9. Balzar, R., L,D. Erman, P. London, and C. William. Hearsay-III: A Domain Independent Framework for Expert Systems. *proceedings, First Annual National Conference on Artificial Intelligence , pp. 108-110*, 1980.

10. *Macsyma Reference Manual*, The Mathlab Group, Laboratory for Computer Science, MIT, December 1978.

11. Mann, W. Text Generation. *American Journal of Computational Linguistics, Vol. 8, No. 2, pp.62-69.* April-June 1982.

12. Buchanan, B.G., and E.H. Shortliffe. *Rule-Based Expert Systems, The MYCIN Experiments of the Stanford Heuristic Programming Project.* Addison-Wesley, 1984.

13. Carr, B. and I.P. Goldstein. Overlays: A Theory of Modelling for Computer Aided Instruction. *MIT AI Memo 406, Memo 40,* February, 1977.

14. Goldstein, I.P. The Genetic Graph: A Representation for the Evolution of Procedural Knowledge. *In Intelligent Tutoring Systems (pp. 51-77), Sleeman and Brown Ed.,* Academic Press, 1982.

15. Waterman, D.A., R. Hayes-Roth. *Pattern-Directed Inference Systems.* Academic Press, 1978.

16. Duda, R., J. Gashing, and P. Hart. Model Design in the Prospector Consultant System for Mineral Exploration. *in Expert Systems in the Microelectronic Age, Michie, D., ed.,* University of Edinburgh, 1979.

17. McDermott, J. R1: An Expert System in the Computer Systems Domain. *1st Annual National Conference on AI,* 1980.

18. Brown, J.S., R.R. Burton, and J. DeKleer. Pedagogical, Natural Language and Knowledge Engineering Techniques in SOPHIE I, II and III. *In Intelligent Tutoring Systems (pp. 227-282), Sleeman and Brown Ed.,* Academic Press, 1982.

19. Senturia, S.D., and B.D. Wedlock. *Electronic Circuits and Applications.* John, Wiley & Sons, Inc., New York, pp. 55-60, 1975.

20. Tversky, A., and D. Kahneman. The Framing of Decisions and the Psychology of Choice. *Science, 211, pp. 453-458,* 1981

21. Burton, R.R., and J.S. Brown. An Investigation of Computer Coaching for Informal Learning Activities. *In Intelligent Tutoring Systems (pp. 79-98), Sleeman and Brown Ed., Academic Press,* 1982.

22. Andrew, Users manual for Prototype Workstation, Release 1, Information Technology Center, Carnegie Mellon University, 1985.

23. Sotoway E., Woolf B., Rubin E., Bonar J. and Johnson, W. L. , MENO II: An Al based programming Tutor, *Journal of Computer-based Instruction, Vol.* 10,1,1983.

24. Johnson, W. L. and Soloway, E., PROUST: A Knowledge Based Program Debugging, *Proceedings of the Seventh International Software Engineering Conference,* pp. 369-380,1984.

25. Morris, J. H., et al, ANDREW: A distributed Personal Computing Environment, *Communication of the ACM,* Vol 29,3, March 1986.

26. Slater, H. S., Petrossian, R. B. P., Shyam-Sunder, S., An Expert Tutor For Rigid Body Mechanics: Athena Cats - Macavity, *Proceedings of the Expert Systems in Government Symposium,* IEEE/CS, McLean, VA October 23-25,1985.