AC 2012-4437: AUTOMATED PROBLEM AND SOLUTION GENERATION SOFTWARE FOR COMPUTER-AIDED INSTRUCTION IN ELEMENTARY LINEAR CIRCUIT ANALYSIS

Mr. Charles David Whitlatch, Arizona State University
Mr. Qiao Wang, Arizona State University
Dr. Brian J. Skromme, Arizona State University

Brian Skromme obtained a B.S. degree in electrical engineering with high honors from the University of Wisconsin, Madison and M.S. and Ph.D. degrees in electrical engineering from the University of Illinois, Urbana-Champaign. He was a member of technical staff at Bellcore from 1985-1989 when he joined Arizona State University. He is currently professor in the School of Electrical, Computer, and Energy Engineering and Assistant Dean in Academic and Student Affairs. He has more than 120 refereed publications in solid state electronics and is active in freshman retention, computer-aided instruction, curriculum, and academic integrity activities, as well as teaching and research.

Automated Problem and Solution Generation Software for Computer-Aided Instruction in Elementary Linear Circuit Analysis

Abstract

Initial progress is described on the development of a software engine capable of generating and solving textbook-like problems of randomly selected topologies and element values that are suitable for use in courses on elementary linear circuit analysis. The circuit generation algorithms are discussed in detail, including the criteria that define an "acceptable" circuit of the type typically used for this purpose. The operation of the working prototype is illustrated, showing automated problem generation, node and mesh analysis, and combination of series and parallel elements. Various graphical features are available to support student understanding, and an interactive exercise in identifying series and parallel elements is provided. When fully developed this engine will be incorporated into a tutorial system designed to supplement conventional instructional approaches.

1. Introduction

1.1. Motivation

Basic linear circuit analysis is frequently a fundamental engineering core course requirement for electrical engineering nonmajors, as well as for electrical engineering majors, and is therefore a very widely taught subject. For electrical engineering majors, the skills learned in this course are essential for their success in subsequent courses. A bad or unsuccessful learning experience may cause students to change their major or drop out of engineering altogether. Traditional lecture-based instruction uses a "one size fits all" approach that fails to adapt to the widely varying learning styles and backgrounds of the students. The goal of this project is to develop computer-aided instruction tools to increase the student success rate in this course by adapting to the needs of individual students. Such tools could be used in a wide variety of ways to supplement either traditional lecture methods or various interactive learning strategies.

The difficulties students encounter while mastering basic linear circuit analysis, in our opinion, often result from a failure to incorporate sufficient active and/or cooperative learning activities in the course, a lack of immediate and effective feedback in homework assignments, and an insufficient number of textbook examples designed to gradually increase in difficulty. Also, errors are sometimes present in textbooks in the worked examples, the problem answers, and the worked solutions, in our experience, as a result of the complexity of generating and solving linear circuit problems by tedious manual methods. Such errors can be very frustrating to students who depend on the correctness of their textbooks, causing them to waste study time and even give up trying to learn at times. Our computer-aided instruction tools are designed to remediate these problems by instantly generating example linear circuit problems that are tailored in size and complexity to an instructor's requirements and preference, solving those problems by the methods typically taught in elementary textbooks, automatically checking student solutions, and displaying detailed solution steps with the correct numerical answers.

Another issue is that textbook solution manuals and old homework solutions are widely available on the Internet, which can encourage students to merely "copy" their solutions from these manuals. Computer-generated problems can be new and unique to every student, greatly reducing incentives for behaviors that bypass the learning process.

Another key pedagogical issue is that most textbooks (and also perhaps, instructors) do not recognize or address the typical misconceptions new engineering students have about electricity and electrical circuits when they enter the course. The need for instructors to address pre- or misconceptions has become broadly known in the physics education community,[1,2] but remains less familiar to most engineering faculty. Such faculty are often not familiar with the extensive literature on student misconceptions related to electricity and circuit analysis,[1,3-11] or else assume (often incorrectly) that such ideas have been "rooted out" in the elementary physics classes. Our own experience administering questions from concept inventories such as DIRECT[1] to students in circuit analysis classes, as well as that of others,[3,4] strongly suggests otherwise. Our plan is to use our computer-aided instruction tools to help remove a number of typical student preconceptions by providing tutorial sequences that require students to correctly master selected basic circuit concepts such as identifying whether circuit elements are in series or parallel, without deriving circuit solutions.

1.2. Background

Various prior projects have used computer-aided instruction in courses on linear electrical circuit analysis.[12-41] The approaches have involved developing intelligent testers and/or tutors, automated rapid grading of homework and/or quizzes, which are sometimes algorithmic in nature (i.e., the numerical values are varied for each student); expert-based advice on selection of optimal circuit-solving strategies; development of multi-media modules for circuit instruction; visualization tools to illustrate circuit behavior; introduction of design topics; use of symbolic circuit analysis programs; and creation of an interactive virtual classroom. The pioneering CircuitTutor software developed by Oakley[18,19,25] was even successfully commercialized for a time through Addison-Wesley,[42] but is now unfortunately obsolete and out of print. While all of these contributions provide useful insights and ideas, many have been only partial prototypes, have covered only limited topical areas, and have very rarely included a facility for automatic problem generation.[32,33] The impact on student learning has not been rigorously established in most cases.

Sustained commercialization of computer-based tools used to teach basic linear circuit analysis has only been achieved to date through textbook publishers, who frequently create elaborate websites with software tools to support their textbooks. Typically, these sites include electronic versions of the textbooks, banks of pre-defined problems from the textbook, (possibly) additional problems, additional worked examples, and some algorithmic problems with the element values varied for every student. The websites may also offer software modules to assign and grade homework with an online grade book and an electronic version of the solution manual. Sometimes, PowerPoint slides, artwork, animations, videos of textbook problems being worked, and multi-part problems with hints and hyperlinks to relevant sections of the textbook are available. Generally, however, detailed step-by-step interactive tutoring, automatic problem or solution generation, and automated grading of responses other than numerical or multiple-choice

answers are not available. And, usually, there is no provision for students to enter circuit diagrams or draw waveforms in a form that can be automatically graded or assessed.

While these web sites are useful adjuncts in teaching the course, they do not offer instruction that can be tailored to the needs of individual students, directly address student misconceptions, or permit significant tutorial interactions. Their principal advantage is that they are supported by the publishers while the relevant book remains in print (typically through many editions), and are therefore sustainable. Moreover, they benefit from the extensive marketing and distribution systems of the publishers. The relevance of such software appears to be increasing steadily. The goal of the present project is therefore to produce software tools that can eventually be integrated into publisher-based systems, providing more in-depth instruction.

1.3. Project Overview

The initial phase of this project is focused on the development of a circuit problem and solution generation engine, which will later be incorporated into a full tutorial system with a variety of interface modules to accept and analyze various forms of student input. The engine is designed to provide an unlimited supply on demand of circuit problems of specified complexity, where both the topology and circuit element values are randomly selected. Further, the system will generate fully illustrated, step-by-step solutions (not just answers) to these problems, which are free of the errors that frequently plague human-generated solutions. The solution methods will include the heuristics typically taught in elementary circuit analysis courses, which distinguishes our approach from the numerical modified nodal analysis approach taken by programs such as SPICE. This engine can be used in many ways, such as providing problems and solutions to an interactive tutorial interface, generating worked examples for students, providing both examples and problems (with solutions) for possible incorporation into textbooks, providing exercises for use in collaborative learning sessions, generating unique homework sets of uniform difficulty that can be automatically graded for every student in a class, creating examination and quiz problems, and so forth. A programmatic interface is provided so that either a graphical user interface or some other program can request and receive problems (including graphics) from the generation engine.

The initial phase of the project has focused on DC circuits, but the same basic algorithms will later be adapted to transient circuits, steady-state AC (phasor) analysis, and Laplace domain analysis. The existing prototype generates random circuits of specified types to solve and draws them (using a PowerPoint interface for the time being), and performs node and/or mesh analysis, with or without pre-simplifications involving the combination of series and parallel circuit elements. For simplicity, we lay out all of our circuits on a square grid, where the circuit elements lie on the edges of the grid squares (no diagonals). This approach is actually quite general as it can be used to draw any planar circuit, which is the type almost always encountered in textbook problems.

In the following, we describe the basic algorithms underlying our circuit generation and solution engine in some detail, as this problem is rarely discussed in the literature. We are aware of only one other project that has developed automated circuit generation routines, which however do not appear to be guaranteed to provide planar circuits suitable for mesh analysis and which have not been demonstrated to generate actual circuit diagrams.[32,33] We subsequently discuss the

graphical features provided to enhance student learning and illustrate an example of a tutorial approach to help students learn to recognize when elements are (or are not) in series and parallel. This issue is at the root of many typical student errors, in the authors' experience.

2. Circuit and Solution Generation Engine

2.1. Circuit Generation Algorithm

Two general approaches could be taken to randomly create circuits of a desired nature. One can create a layout of a circuit and from that extract the circuit "netlist," which is the abstraction typically used in nodal or modified nodal analysis. The latter consists of a list of the circuit elements, the nodes to which they are connected (where the order of the nodes specifies polarity if applicable), their values, and the specification of control variables in the case of dependent sources. Alternatively, one could create circuits abstractly in the form of netlists or circuit graphs, followed by a procedure to generate corresponding layouts. In this project we selected the former approach. Our rationale is in part that circuits that are topologically equivalent may not always be recognized as such by beginning students, and can therefore be considered distinct for pedagogical purposes. Furthermore, our approach guarantees planar circuits that are suitable for mesh analysis, which may be difficult to ensure in a graph-oriented approach. Also, it is easier to program netlist extraction than a layout algorithm.

As noted above our circuits are always laid out on a square grid of points for simplicity. The grid is defined to consist of a rectangular array of $m \times n$ squares, whose $(m+1)(n+1)$ corners are connection (grid) points and whose $2mn+m+n$ edges (which we denote as segments) may be short circuits (wires), open circuits (blanks), or ideal circuit elements chosen from resistors, capacitors, inductors, independent voltage and current sources, dependent voltage or current sources, or single-pole single-throw switches. (Ideal operational amplifiers and transformers will be added later, along with mutual inductances, but those cases require special approaches not discussed here.) Neglecting switches for the moment (needed only for transient circuits), there are therefore nine possible items that can be placed on each of the segment (including shorts and opens). Note that a node can consist of multiple grid points connected by shorts, and meshes may consist of one or more adjacent squares (due to segments that are open circuits).

It might well be thought (by those who have not actually tried to do it) that one could generate circuits simply by randomly "throwing" such items onto the grid, and then examine the resulting circuits and throw away those that are not "valid" problems for students (based on criteria that are defined below). The problem is that for all but the very smallest circuits, the number of possibilities that must be examined becomes incredibly large and only a tiny fraction is valid, rendering this approach completely unworkable (even if problems are generated in advance and stored). The problem is much like waiting for the proverbial monkey to type one of Shakespeare's plays by random chance. Even for a very small case with $m=2$ and $n=1$ and therefore 7 segments, $7^9 \approx 4.8\text{x}10^6$ circuits are possible (where element values are not yet selected).

2.2. Stepwise Approach to Circuit Generation

To make this problem more tractable, we use two basic strategies. First, we generate circuits by a series of three successive steps. (The second strategy, discussed in later sections, involves using techniques to ensure that the results of each generation step are automatically valid to the extent possible.) In the first circuit generation step, we place only open and short circuit elements on the grid, with the understanding that open circuits will always remain open, but short circuits may either remain as such or be replaced by other circuit elements (but not opens). We call the resulting grid of lines [illustrated in Fig. 1(a)] a circuit topology. This topology defines the ultimate number of meshes in the circuit, the number of "active" points in the grid (that have one or more connection to them), and the number of "active" segments (that are not merely open circuits). For m=2 and n=1, there are now only 7^2 =49 such possible topologies. "Invalid" topologies are rejected, and only valid ones can be used in the second step (validity is discussed below).

In the second step, some of the active segments in the topology are replaced ("populated") by generic circuit elements (represented as boxes), as shown in Fig. 1(b). The number so replaced in this "populated topology" is the desired number of nontrivial circuit elements in the circuit (i.e., *branches* in the terminology of network theory). The generic elements are all later replaced by specific circuit elements, but shorts that remain will stay as shorts. The number of possibilities is now only the number of active segments raised to the power of two, and thus still modest. The populated topology now has specific numbers of nodes and branches as well as the set number of meshes established in the topology generation step.

In the third step, the generic elements are all replaced by specific circuit elements, based on the desired number of each in the circuit. Each of those elements is assigned a randomly chosen value (within a user-specified range), and control variables are selected for dependent sources as the currents or voltages of other branches in the circuit. A sample result is shown in Fig. 1(c). The detailed procedure is designed to ensure circuit validity as discussed below.

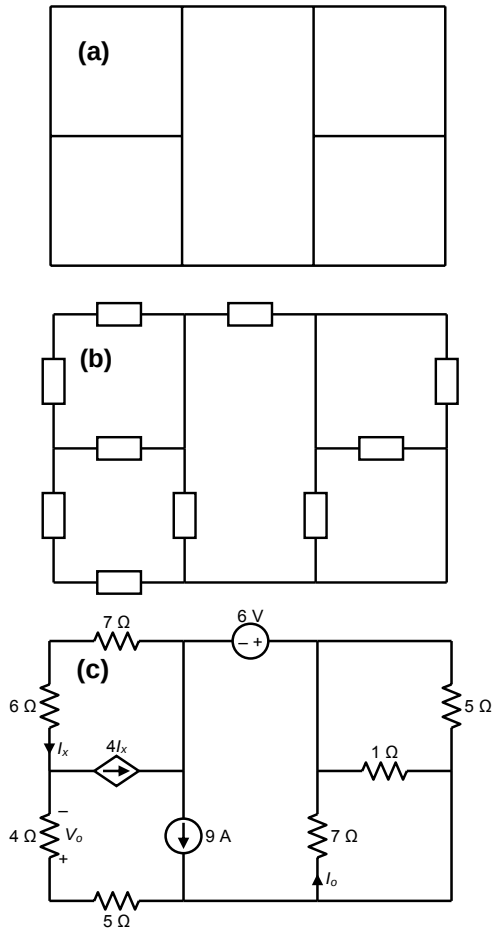

Fig. 1. The three stages of the circuit generation process. (a) Circuit topology (line segments). (b) Topology populated with generic circuit elements. (c) Placement of actual circuit elements. The student is asked to compute I_o in this problem.

2.3. Strict Requirements for Valid Circuits

Circuit problems are considered unacceptable if they either 1) have no valid, unique solution (insoluble), or 2) are not of a type that is typically assigned to

students in an introductory linear circuit course, as exemplified by problems in the major textbooks used in this area. Some exceptions may be made in the second case for occasional problems assigned for specific pedagogical reasons, but not routinely. Criterion 1 is discussed in this section and Criterion 2 in the next section.

Insoluble problems occur when either independent or dependent sources are arranged in an inconsistent fashion, or circuits containing dependent sources have no valid solution (which typically occurs for specific values of gains along with specific element values). An example of the former (inconsistent) case is two or more current sources in series with different current values, or two or more non-identical voltage sources in parallel. One of these sources might be a dependent source, in which case the circuit is inconsistent unless the control variable is forced to have some very specific value (which would be of purely pedantic interest). For this purpose, we include short circuits as being ideal voltage sources whose voltage is zero, and open circuits as ideal currents sources whose current is zero. I.e., a current source cannot be in series with an open circuit or a voltage source in parallel with a short circuit, as Kirchoff's current law (KCL) or voltage law (KVL) would be violated, respectively. Furthermore, for DC or transient circuits we must consider capacitors to be open circuits and inductors to be short circuits in this prohibition (transient circuits must be consistent when they are in steady-state).

We must actually generalize the restrictions on series and parallel sources to ensure that neither KCL nor KVL is violated and that all node voltage have uniquely determined values. No star (set of elements all connected to a common node) should consist entirely of independent or dependent current sources (which include open circuits, and capacitors in the DC case) whose values result in a violation of KCL. This restriction generalizes the case of sources in series, which involves one or more stars of only two elements. (Application of this rule will automatically take care of elements that are in series with elements in a branch of the star, since they introduce other nodes to which the same rule is separately applied.) Even if the sources satisfy KCL, the central node in the star has an indeterminate voltage and the node equations are therefore not uniquely solvable with a single reference node. Similarly, no loop in the circuit should consist entirely of independent or dependent voltage sources (including short circuits and inductors) whose values result in a violation of KVL. This rule generalizes the case of parallel voltage sources, as each set of parallel sources forms a loop. Even if the sources satisfy KVL, the mesh currents will not be uniquely determined in this case. To ensure that both node and mesh equations have unique solutions we must never allow a star of current sources or a loop of voltage sources (including capacitors or inductors in the DC/transient case and open or short circuits in all cases in this prohibition).

A further generalization is actually necessary to guarantee solvability of node or mesh equations. A "deactivated" or "dead" circuit (in which all independent voltage sources are replaced by short circuits and all independent current sources are replaced by open circuits, i.e., turned off) must be fully connected for the node equations to have a unique solution. This statement is proven rigorously, e.g., by Davis.[43] It generalizes the requirement on stars discussed above, since the node at the center of the star will be isolated in the dead circuit if the star consists entirely of current sources. Similarly, the mesh equations have a unique solution (unique set of mesh currents) only if the corresponding dead circuit is "coupled," meaning that one can pass from any mesh (including the exterior mesh) to any other by passing over a shared resistor.[43] Any loop consisting only of shorts and voltage sources will cause this test to fail.

Unique solvability of both node and mesh equations is readily achieved by placing all inductors and voltage sources on the twigs of some valid tree of the circuit, and all capacitors and current sources on the links of that same tree. By definition, a tree is a set of branches (called twigs) that connects all nodes but does not contain any loops.[44] Deactivating all sources cannot leave a node isolated and result in a disconnected circuit if the sources are placed as specified. Moreover, the deactivated circuit consists only of twigs in this case, so there cannot be any decoupled meshes (there are no loops at all!). For steady-state sinusoidal AC circuits, the placement of the capacitors and inductors does not need to be restricted (as long as we avoid zero frequency), only the current and voltage sources.

Situations where inconsistencies occur due to specific values of dependent source gains and element values are more difficult to avoid. As this situation occurs rarely, we find that simply rejecting circuits where the node equation matrix is singular and re-starting the circuit generation routine in this case is an entirely satisfactory approach. The re-starting is performed a limited number of times but very few re-starts are typically necessary.

2.4. Desired Optional Features for Most Circuits

The above approach guarantees a circuit that can be solved. Other optional criteria that we usually wish to fulfill, to make the problems of a type that is typically assigned are as follows:

1. The graph of the circuit should be fully connected. (The existence of a valid tree as assumed above is only valid for a connected circuit.) Circuit graphs consisting of multiple unconnected sections usually reduce to multiple independent problems (unless they are linked by dependent sources whose values in one circuit section depend on control variables in another section), and there appears to be no point in creating such problems. This criterion can be applied in the topology generation step; a connected topology cannot yield an unconnected circuit.

2. In addition to being connected, circuits should normally not be "hinged." A hinged circuit is one that can be drawn in a way such that two or more sections are connected to each other by only a single wire or circuit element.[45] Since KCL implies that the current through such a wire must be zero, the different portions are essentially isolated (except for a common potential reference) and amount to multiple independent problems (except again when linked by dependent sources). Simple transistor models (without external circuitry) are often hinged, so that one might occasionally wish to allow such circuits, but not as a routine matter. Some topologies are hinged [as in Fig. 2(a)], and are rejected, as they will always yield hinged circuits. A special case of a hinged topology is one where a circuit element is "dangling," or not connected on both ends.

Other topologies become hinged only after they are populated in particular ways by generic elements [Fig. 2(b)], so that one must inspect for hinging both before and after the generic population step. Unhinged populated topologies cannot become hinged during element placement. A hinged topology can be recognized by testing if it turns into an unconnected circuit when a single grid point (and the segments attached to it) are removed. A hinged populated topology is most easily tested by forming the associated netlist and determining if the circuit so represented becomes disconnected after removing any node and the elements

connected to it, or if any element is connected on both ends to the same node (shorted). Shorted elements obviously play no useful role in the circuit.

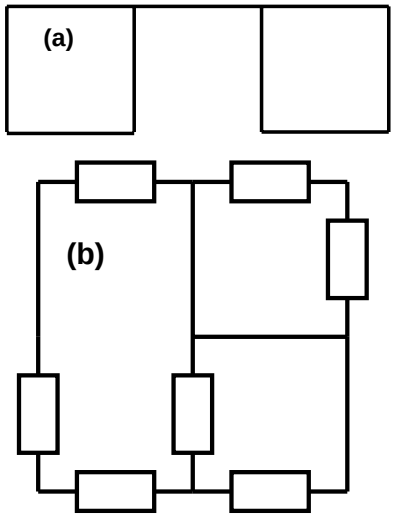

Fig. 2. Examples of (a) a hinged topology; and (b) a topology that became hinged only after population with generic elements. The two elements at upper right are effectively isolated from the remainder by a path of shorts (the circuit could be re-drawn such that the two parts are connected by a single wire).

3. Circuit elements that are redundant and serve no useful purpose should normally be avoided. Such elements occur whenever a star has only one element that is not a current source or a capacitor (the latter for DC circuits). The current through this element (whether it is a voltage source or a passive element) is then fixed by KCL, so that element itself has no useful function (except possibly in practice, e.g., a series resistor used to monitor the output of a current source). A special case is a voltage source in series with a current source, where the voltage source serves no useful function. Similarly, any element that is the only one in a loop that is not a voltage source (or inductor, for DC circuits) has a voltage fixed by KVL and is also redundant, whether it is a current source or a passive element. A special case is a current source in parallel with a voltage source, where the current source serves no useful function.

In other words, all stars should have at least *two* elements that are not current sources or capacitors, and all loops should contain at least two elements that are not voltage sources or inductors. These conditions are more restrictive than the conditions stated above to avoid inconsistencies (that at least *one* element of the specified type should be present in each star and loop). We enforce this rule when placing circuit elements, by an algorithm that rules out placements that would violate it. We provide user-selected options to avoid all redundant elements, to avoid redundant sources but allow redundant passive elements, or to permit all redundant elements. A fully rigorous test for redundancy would also disallow any situation where the corresponding dead network is either unconnected or hinged (a generalization of the test discussed above for the solvability of the node equations). Equivalently, any cutset (including stars as a special case) would have to contain at least two resistors. To date we have not enforced this additional restriction.

4. In general voltage sources in series with each other and current sources in parallel with each other are allowed, but given that they could be combined to form a single equivalent source, we provide an option to limit the maximum number of each to any value ≥ 1. These restrictions are enforced when placing circuit elements.

5. We prohibit meshes in our layouts that consist entirely of wires (shorts), as they serve no useful function. Moreover, allowing such a case would reduce the actual number of meshes in the topology to a smaller number after populating it with generic elements, so that the desired

number of meshes would no longer be achieved. This restriction is enforced during population with generic elements by insisting that each mesh contain at least two circuit elements. A mesh containing only one circuit element would automatically have a shorted element and therefore be hinged. (Loops that consist entirely of shorts and that are not meshes are automatically prohibited because they would make the circuit hinged by isolating the portion inside the loop from that outside it.)

6. A circuit layout on a grid of $m \times n$ squares is not considered acceptable unless the topology extends the full length and width of that grid. If it does not, it should be classified as a proper layout only on a suitably smaller grid. We enforce this rule in our topology generation algorithm. One could also reject populated topologies where two or more rows or columns consist entirely of shorts, as they could be "collapsed" onto a smaller grid and are therefore really of a smaller size than that desired. We have not yet enforced this restriction, however.

7. Typically we prohibit problems in which all node voltages are determined directly by voltage sources, or in which all mesh currents are determined directly by current sources. Such problems are relatively trivial for node or mesh analysis, respectively, and are therefore avoided. They can be allowed in some cases for pedagogical purposes and there is a program switch to do so.

8. Many textbook problems are designed with the bottom row consisting entirely of wires, so that it becomes a natural choice of ground. Such a feature could be incorporated but it does not seem fundamentally important so we have not done so to date.

2.5. Restrictions on Control Variables for Dependent Sources

The vast majority of textbook problems (and practical applications) involve dependent sources whose output is controlled by a current through or a voltage across a passive element rather than by a current through or a voltage across another source, so we enforce that limitation. A dependent current source should not be in series with the branch on which the control voltage or current is defined, as it causes a fundamental inconsistency unless a specific gain is chosen (and is equivalent to a short circuit if the required gain is chosen). Likewise, a dependent voltage source should not be in parallel with a branch on which the control variable is defined, as it would be either inconsistent or equivalent to an open circuit. These statements could be generalized to stars whose branches are all current sources except for one passive element on which the control variable defined, and where one of the sources is the dependent source in question. However, we already prohibit such stars when redundant circuit elements are being avoided as discussed above, so this generalization is generally unneeded. Similar comments apply regarding generalization of the parallel case to loops of mostly voltage sources.

Furthermore, dependent voltage sources that are in series with resistors on which their control variables are defined are equivalent to resistors of certain values (which might be negative), so there seems to be little point in general to allowing such cases. The same comment applies to dependent current sources that are in parallel with resistors on which their control variables are defined. We therefore avoid using any branches that are in series or parallel with any dependent source to provide the control variable for that source. This prohibition is enforced when choosing control variables for dependent sources, which is done after placing all the elements

including those sources in the circuit. Each branch voltage (or those in parallel with it) and branch current (or those in series with it) can be used only once as a control variable.

The above series/parallel restrictions must be relaxed when dependent sources are introduced in single loop or single node-pair circuits, as they are in some textbooks. That case might be considered useful for pedagogical purposes (to introduce the idea of dependent sources using very simple circuits), but practical circuits would never be of this type.

2.6. Topology Generation Algorithm

The simplest strategy to generate a valid topology would be to place wire segments onto a grid at random, and reject any resulting topologies that are disconnected, hinged (including a dangling wire), or do not have the desired number of meshes. We have however designed a much more efficient algorithm to guarantee circuits that are connected, have no dangling segments, completely use the width and length of the pre-defined grid dimensions (m x n squares), and have at least the required number of segments to accommodate the desired number of circuit elements. This algorithm works by selecting the first point at random in the grid, then randomly attaching a (shorted) segment to it in some direction. Then, a point having a connection is picked at random (provided there is room to add another segment connected to it) and a new segment is randomly attached to that point, building a branched structure that may contain loops. This process is repeated until at least the required minimum number of segments has been placed. Then, if there are any dangling segments (i.e., points with only one segment connected to them), additional segments are randomly added by the same process until all dangling segments are eliminated.

Next, if the network does not extend the full length and width of the grid, additional segments are randomly added until it does. Throughout this process, points with only one existing connection are given preferential weighting as the source for a new segment, to avoid creating additional "branching" in the layout. Also, segments that would continue along a straight path instead of turning are given preferential weighting to help “fill out” the topology on a larger grid and create meshes more readily. The weighting factors were empirically optimized. The resulting topologies are not guaranteed to have the correct number of meshes or to be unhinged, but those requirements are checked and the process simply repeats if they are not. A small number of attempts is typically needed.

2.7. Algorithm for Generic Population of a Topology

The second step in circuit generation is to replace some of the wire circuits in the topology with generic circuit elements. We use an algorithm that is designed to minimize the probability that a populated topology will become hinged after population. Any populated topology (layout) having a path of shorts extending from one point on the exterior of the circuit to a different point on the exterior (where the path cuts across the circuit and is not part of the exterior mesh) will cause the layout to be hinged. Also, any mesh that does not contain at least two generic elements (including the exterior “mesh” around the outside of the circuit) will result in either a mesh of shorts if it has zero elements (which we wish to prohibit as explained above) or a single element that is shorted by the remainder of the mesh (which constitutes a hinged circuit). Thus, every circuit with M meshes requires at least M+1 generic elements, where M–1 of them are placed on

segments that are shared between two meshes and the remaining two are placed on exterior segments (part of the exterior mesh and therefore unshared). Some topologies require more elements than this to avoid being hinged (e.g., by a path of shorts), but this is a minimum.

The algorithm proceeds as follows. We first place an element on every segment that directly connects two points on the exterior mesh (and that is not itself part of the exterior mesh). We then randomly place elements on shared segments until $M-1$ total have been placed on such segments (but with no more than two per mesh). Elements are then placed on two randomly chosen exterior segments, but again placing no more than two per mesh. At this point, additional elements are placed on any mesh that does not already have two elements. Finally, any additional elements that need to be placed are placed randomly anywhere. A netlist is then extracted for the resulting layout, and tested and rejected if it is found to be hinged (but this happens very rarely using this approach). In case of rejection the process is repeated.

2.8. Algorithm for Element Placement

The populated topology is next examined to identify all elements in series and parallel, which can be determined at this stage. As our element placement algorithm is based on trees and cotrees of the circuit as discussed above, we use a recursive algorithm similar to that described by Grimbleby[46] to find all or many of the possible trees for the populated topology. One of these trees is selected at random. The twigs of the tree are then populated randomly by the specified numbers of both independent and dependent voltage sources and inductors, using a complicated algorithm that limits the number of voltage sources in series to a specified value and avoids most redundant sources and passive elements as discussed above (if the user so specifies). This process converts formerly generic elements into the desired types. The links of the tree are then populated by the specified numbers of independent and dependent current sources and capacitors using a similar algorithm to limit the number of current sources in parallel and to again prevent most redundant sources and passives if so desired. Next, the remaining generic elements are converted to resistors. Element values and polarities are randomly chosen in user-specified ranges. After all elements are placed, the control voltages or currents are randomly selected for each dependent source, respecting the rules mentioned above. If the specified restrictions on series/parallel sources and redundant elements cannot be satisfied for a given tree, a different tree is picked until all available ones have been tried. If the placement algorithm still fails, new generic populations of the topology are tried and failing that, new topologies.

2.9. Sought Quantity Selection

Most circuit problems ask the student to compute one (or a few) specified voltages, currents, and/or powers in a circuit. The voltage (differences) are often voltages across a specific element (other than an independent voltage source, of course!), which we term branch voltages. Alternatively, they are sometimes voltages between nodes that are not directly connected by a circuit element, which we call non-branch voltages. The currents may be either currents through a specific branch that is not an independent current source (termed branch currents), or rarely, currents through specific wires that are internal to a node (and are therefore dependent on a specific layout), which we term non-branch currents. The power(s) may be that either absorbed or supplied by an independent or dependent source or by a passive element. These unknown values that the student is asked to find are termed the "sought quantities" here. In some

problems, some or all of the node voltages might instead be requested (referenced to a specified ground), or some or all of the mesh currents. Further, some special types of problems ask students to find the value of a circuit element that will cause some condition to be fulfilled, such as causing a certain voltage or current to exist at some point, maximizing power dissipation in a resistor, and so forth.

Our program randomly selects the sought quantities in the circuit based on user specifications of how many variables of each type should be requested (though we do not currently support non-branch currents). We however prohibit sought voltages from being those of an independent voltage source or any element in parallel with one (which would have trivial answers) and likewise prohibit sought currents that are those of an independent current source or are in series with one. We also avoid specifying the power of an element that is in series or parallel with an independent current or voltage source, respectively. Sought currents are not allowed to be in series with any other sought or control currents for dependent sources, to avoid redundancy, and sought voltages are not allowed to be in parallel with any other sought or control voltages. Control currents and voltages can however be sought quantities. Element values are not yet supported as unknowns but may be in the future.

2.10. Circuit Specification Approaches

When specifying the desired type of circuit problem to generate, possible quantities to specify are the number of squares in the grid ($m \times n$), the number of each type of circuit element to use, the number of nodes (N), and the number of meshes (M). However, these values cannot all be specified independently. For a connected, unhinged circuit having B total circuit elements (branches), the well known requirement[43] is that

$$N = B - M + 1. \tag{1}$$

The number of twigs on a tree (T) is given by $T = N - 1$, so that the number of links L (i.e., branches that are not twigs) is $L = B - T = M$ based on Eq. (1). A valid circuit must have at least two nodes, so that $M \leq B - 1$. We denote the numbers of (both independent and dependent) voltage and current sources, resistors, capacitors, and inductors as N_V, N_I, N_R, N_C, and N_L, respectively, so that $B = N_V + N_I + N_R + N_C + N_L$. To be able to place voltage sources and inductors on twigs in DC and transient circuits, we need $N \geq N_V + N_L + 1$ and therefore $M \leq B - N_V - N_L$ based on Eq. (1). To place current sources and capacitors on links in DC and transient circuits, we need $M \geq N_I + N_C$ and therefore $N \leq B + 1 - N_I - N_C$ based on Eq. (1). For steady-state AC circuits the inductors and capacitors can be placed anywhere, so that the corresponding constraints would instead be that $N \geq N_V + 1$, $M \leq B - N_V$, $M \geq N_I$, and $N \leq B + 1 - N_I$.

There are also practical constraints based on the size of the rectangular layout grid, which has $m \times n$ squares, $(m+1)(n+1)$ grid points, and at most $2mn + m + n$ segments where elements can be placed. Since each mesh requires at least one square on the grid, we need $M \leq mn$. Each node requires at least one grid point, so that $N \leq (m+1)(n+1)$. Each branch requires at least one segment, so that $B \leq 2mn + m + n$. In practice we can satisfy these constraints by increasing m and n as needed, but if the user wishes to restrict the grid size, or it is limited by program constraints, then the numbers of nodes, meshes, and elements must be limited.

Some additional optional considerations are that first, there should be at least one independent source in most problems (except those asking for an equivalent impedance or resistance), so $N_V + N_I \geq 1$. A useful circuit also has to have some way to dissipate the energy supplied by the sources, so one would also usually require $N_R \geq 1$. Furthermore, if we wish to avoid having all node voltages being directly determined by voltage sources, we need $N \geq N_V + 2$, and to avoid all mesh currents being determined directly by current sources, we need $M \geq N_I + 1$.

The present user interface offers the user several different options for specifying circuits (with more to be added later). The user can specify all of the numbers of different types of elements to be used, the number of meshes, and the grid size, in which case the number of nodes is automatically adjusted to satisfy Eq. (1) and the other constraints mentioned above. Alternatively, the user can specify all of the numbers of elements to use, the number of nodes, and the grid size, in which case the number of meshes is automatically adjusted to satisfy Eq. (1) and the other constraints. A third option is to specify the numbers of sources and reactive elements, the numbers of nodes and meshes, and the grid size, in which case the number of resistors is automatically adjusted to satisfy Eq. (1) and the other constraints. In these three options the grid size is increased automatically up to a limit of $m = n = 5$ as needed (though most circuit problems will of course be much smaller). A variety of other options will be added at a later date. Special procedures are of course needed for the case of circuits with switches, op-amps, transformers, and mutual inductances, which will be added as the project progresses.

The tutorial system will include special code to ensure that feasible combinations of parameters are specified when it requests problems from the circuit generation engine.

2.11. Solution Algorithms

Once valid problems have been created, the program is designed to generate fully worked solutions to each problem using a variety of the techniques typically taught in introductory circuit analysis courses, such as voltage and current division, node and mesh analysis (using supernodes and supermeshes as needed), combination of elements in series and parallel as appropriate (perhaps prior to node and mesh analysis or other techniques), superposition, source transformation, and use of Thévenin and Norton equivalent circuits to represent part of the circuit. The algorithms used for these solutions are just those described in common introductory textbooks,[43,47-52] and are therefore not discussed here in detail. The solutions include successive re-drawings of the circuit diagram as needed, display of the relevant equations, the matrix form of the equations where needed, and the actual numerical answers for the specified sought quantities. Re-drawing the circuit diagrams in the case of wye-delta transformations and source transformations may require expanding the original grid and/or "splitting" the original diagram to provide room for the new elements. Automatically generated text provides an explanation of what is being done at each step.

The actual procedures used for many of the above techniques depend on the particular quantities being sought in the problem. An element whose voltage is sought should not be combined in series with other elements, for example, and likewise an element whose current is needed should not be combined in parallel with other elements. Wye-delta transformations must be prohibited for elements used for sought quantities. We also avoid any combinations or transformations that

would eliminate a control variable used for a dependent source. Any transformation that eliminates a node specified in a sought non-branch voltage is also prohibited.

To date, we have completed the code for node and mesh analysis and for combining series and parallel elements (all for DC circuits lacking reactive elements). An example of a problem being solved in successive steps and finally by nodal analysis (after presimplification) is shown in Fig. 3. Another example of a problem being solved by mesh analysis without presimplification is

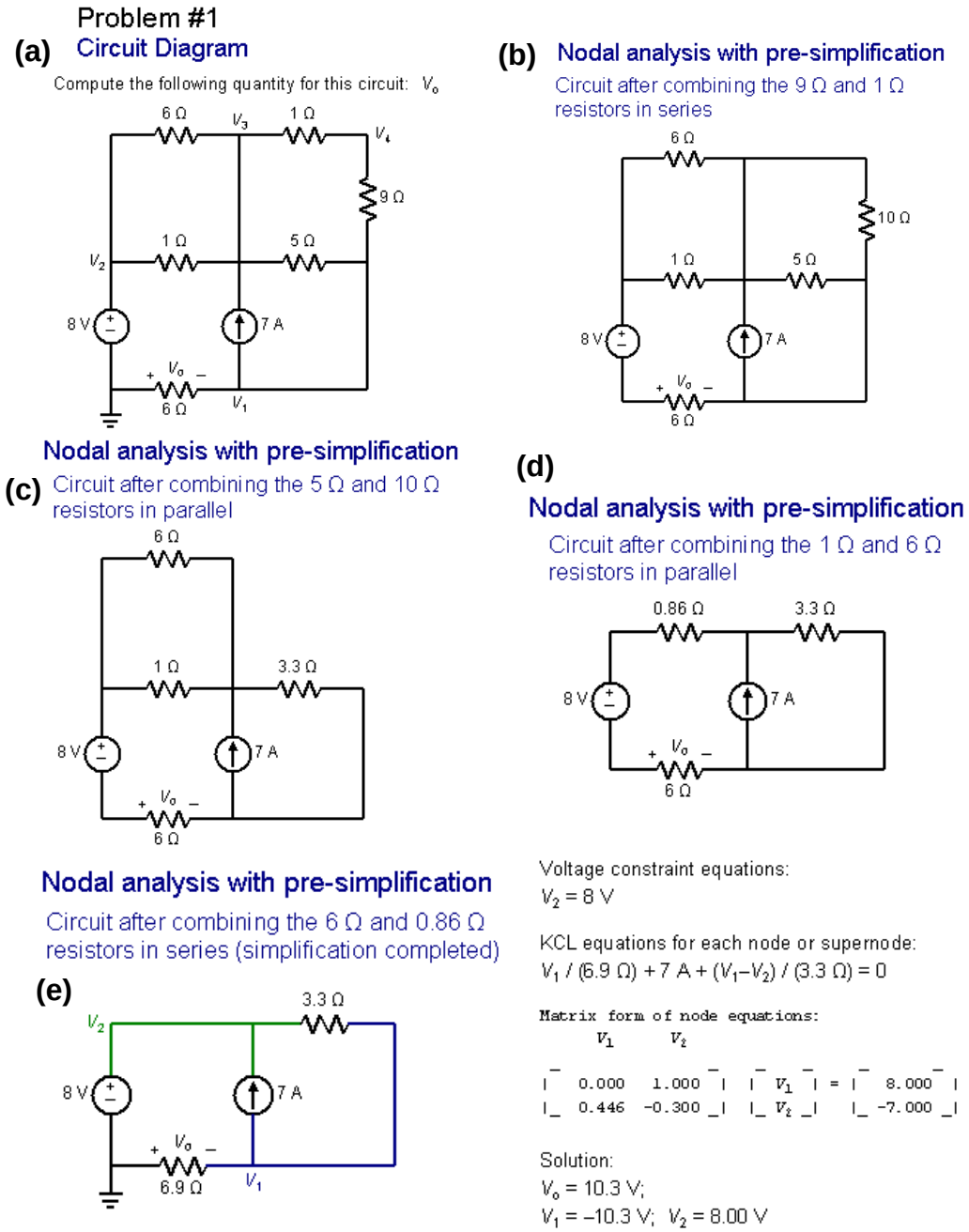

Fig. 3. Automatically generated step-by-step solution of a circuit problem using series and parallel combinations followed by nodal analysis. Steps (a)-(e) in the process are each illustrated in a separate automatically annotated drawing.

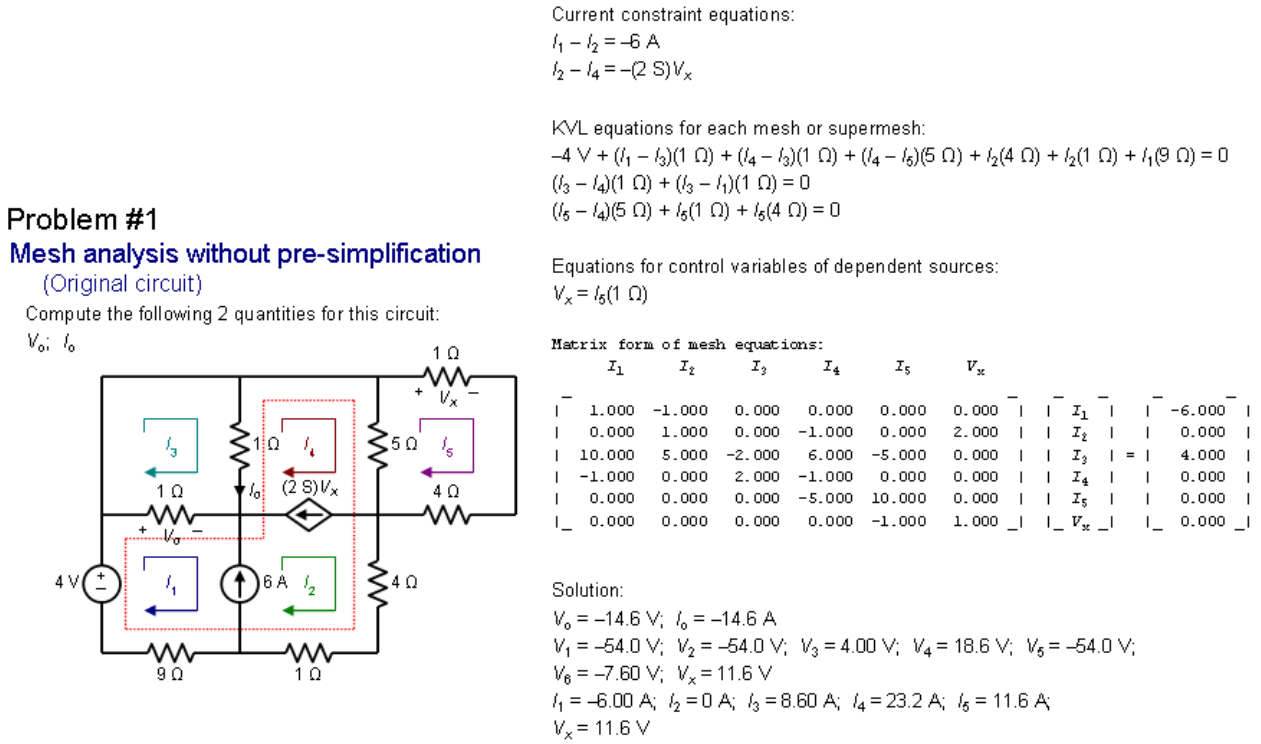

Fig. 4. Example of automatically generated mesh analysis without pre-simplification. Mesh currents are labeled by colored loops and the supermesh path by a dashed loop.

shown in Fig. 4. As the project progresses we plan to add support for all of the other solution methods as well as transient, steady-state AC (phasor), Laplace, and op-amp based circuits. We also plan to add support to check solutions input by students against those generated by the program (including circuit re-drawing, equations, and numerical answers).

3. User Interface Design and Features

3.1. Problem Specification Interface

At present users specify the types of problems and solutions to be specified through an interface implemented in Visual Basic for Applications. One tab of the dialog box is illustrated in Fig. 5. The first tab is used to specify the type of circuit (only DC at present) and the manner in which the circuit is specified (from a drop-down menu), and the other boxes are used to specify the numbers of elements, nodes, meshes, and grid size (though

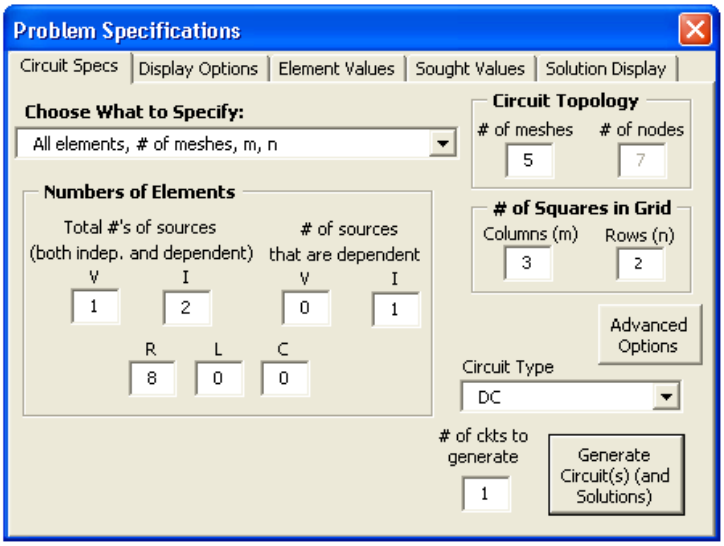

Fig. 5. Screen shot of the user interface dialog box (first tab shown).

not all can be set by the user in a given specification method as discussed in Section 2.10). Code behind the dialog box is used to validate all entries and adjust other parameters (with appropriate warnings) as needed. An advanced options box is used to specify the maximum number of voltage sources in series, current sources in parallel, whether redundant sources and/or passive elements are allowed as discussed in Section 2.4, and whether circuits whose node voltages are all determined directly by voltage sources and circuits whose mesh currents are all determined directly by current sources are allowed. The Sought Quantities tab (not shown) is used to specify the types and number of variables for which the student will be asked to solve, as discussed in Section 2.9.

3.2. Display Options and Features

Various display options have been developed to eventually support the use of the circuit diagrams in a tutorial setting. For example, circuit elements can be labeled either with their values (such as 4 Ω) or with their names (such as R2). The different nodes can be automatically colored with different colors to help students recognize them [see Fig. 3(e)] and the actual element symbols can even be blanked out, leaving just the wires and leads to emphasize the nodes. Groups of elements that are in series or parallel can be highlighted red on request. The choice of reference node can be changed to any specified node, in which case the node numbering and all equations are automatically updated to match. Mesh currents can be labeled automatically on arbitrarily-shaped meshes, as can the loops that constitute supermeshes (see Fig. 4). Another feature is that the currents leaving a user-selected node or supernode can be automatically labeled with colored arrows, to help students understand the origin of each term in the KCL equation for that node or supernode. We plan to optionally color each term in a KCL equation to match the color of the corresponding current arrow. A similar approach will be used for each term in a KVL equation and the corresponding voltage drop around a loop. A close graphical correspondence between the diagram and the equations can thus be achieved. We have also implemented an interactive exercise in which students are asked to list all sets of elements in a circuit diagram that are in series or parallel, providing feedback on their answers as well as keeping "score." This exercise has already been used successfully in an introduction to electrical engineering class, where students were required to complete it a given number of times and until they achieved a target score.

4. Conclusion

The progress to date in developing a circuit generation and solution engine has been described, including details of the underlying algorithms. The criteria to use in generating "typical" textbook-like problems have been considered in detail. Future work will focus on expanding the capabilities to treat all common textbook solution methods and heuristics, and extension to phasor analysis, transient circuits using a differential equation approach, Laplace domain analysis, and various special cases including op-amp circuits. Additional software components to accept and analyze student input in various forms, carry out scripted tutorial sequences that make use of the generation and solution engine, and report student progress to the instructor are planned for future work.

Acknowledgment

This work was supported by the National Science Foundation through the Transforming Undergraduate Education in Science, Technology, Engineering and Mathematics Program under Grant No. DUE-1044497.

References

[1]P. V. Engelhardt and R. J. Beichner, "Students' understanding of direct current resistive electrical circuits," Am. J. Phys. **72**, 98 (2004).
[2]D. Hestenes, M. Wells, and G. Swackhamer, "Force Concept Inventory," Phys. Teach. **30**, 141 (1992).
[3]R. Streveler, M. Geist, R. Ammerman, C. Sulzbach, R. Miller, B. Olds, and M. Nelson, "Identifying and investigating difficult concepts in engineering mechanics and electric circuits," in *Proc. 2006 Annual Mtg. Amer. Soc. for Engineering Education* (2006), p. 2006.
[4]D. Holton, A. Verma, and G. Biswas, "Assessing student difficulties in understanding the behavior of ac and dc circuits," in *Proc. ASEE 2008 Ann. Conf. & Exposition* (Amer. Soc. Engrg. Educat., Washington, DC, 2007), p. AC2008.
[5]M. Caillot (ed.), *Learning Electricity and Electronics with Advanced Educational Technology* (Springer-Verlag, Berlin, 1993).
[6]M. S. Steinberg and C. L. Wainwright, "Using models to teach electricity--The CASTLE project," Phys. Teach. **31**, 353 (1993).
[7]R. Duit, W. Jung, and C. von Rhoneck (ed.), *Aspects of Understanding Electricity--Proceedings of an International Workshop* (Verlag, Schmidt, & Klaunig, Kiel, Germany, 1984).
[8]D. Sokoloff, The Electric Circuits Concept Evaluation (ECCE), http://physics.dickinson.edu/~wp_web/wp_resources/wp_assessment.html#ECCE.
[9]R. Cohen, B. Eylon, and U. Ganiel, "Potential difference and current in simple electric circuits: A study of students' concepts," Am. J. Phys. **51**, 407 (1983).
[10]M. S. Steinberg, "Reinventing electricity," in *Proc. Internat. Seminar on Misconceptions in Science & Mathematics*, edited by H. Helm and J. Novak (Cornell Univ., Ithaca, NY, 1983), p. 388.
[11]L. C. McDermott and E. H. van Zee, "Identifying and addressing student difficulties with electric circuits," in *Aspects of Understanding Electricity*, edited by R. Duit, W. Jung, and C. von Rhoneck (Verlag, Schmidt, & Klaunig, Kiel, Germany, 1984).
[12]M. Nahvi, "Teaching introductory courses in electrical engineering to engineering majors, new tools and context," in *Proc. 1988 Frontiers in Education Conf.* (1988), p. 76.
[13]M. Nahvi, "A computer based intelligent synthetic tutor-tester for electrical engineering," in *Proc. IEEE Internat. Conf. on Systems, Man, & Cybernetics* (1990), p. 742.
[14]H. E. Hanrahan and S. S. Caetano, "A knowledge-based aid for dc circuit analysis," IEEE Trans. Educ. **32**, 448 (1989).
[15]J. S. Demetry, B. Black, D. Voltmer, M. Nahvi, and J. Jones, "Computer-assisted interactive instruction: Results from a developmental effort," in *Proc. 1992 Frontiers in Education Conf.* (1992), p. 662.
[16]A. Yoshikawa, M. Shintani, and Y. Ohba, "Intelligent tutoring system for electric circuit exercising," IEEE Trans. Educ. **35**, 222 (1992).
[17]F. de Coulon, E. Forte, and J. M. Rivera, "KIRCHHOFF: An educational software for learning the basic principles and methodology in electrical circuits modeling," IEEE Trans. Educ. **1993**, 19 (1993).
[18]B. Oakley II, "Use of the Internet in an introductory circuit analysis course," in *Proc. 1993 Frontiers in Education Conf.* (1993), p. 602.
[19]B. Oakley II and R. E. Roper, "Implementation of a virtual classroom for an introductory circuit analysis course," in *Proc. 1994 Frontiers in Education Conf.* (1994), p. 279.
[20]G. F. Shannon, "Multi-media computer based teaching--A case study," in *Proc. IEEE 1st Internat. Conf. on Multi-Media Engineering Education* (1994), p. 398.
[21]J. R. Jones and D. A. Conner, "The development of interactive tutorials for introductory circuits," in *Proc. IEEE 1st Internat. Conf. on Multi-media Engineering Education* (1994), p. 108.

[22]J. Teng, J. Fidler, and Y. Sun, "Symbolic circuit analysis using Mathematica," Int. J. Electr. Eng. Educ. **31**, 324 (1994).
[23]E. R. Doering, "*CircuitViz*: A new method for visualizing the dynamic behavior of electric circuits," IEEE Trans. Educ. **39**, 297 (1996).
[24]L. P. Huelsman, "Symbolic analysis--A tool for teaching undergraduate circuit theory," IEEE Trans. Educ. **39**, 243 (1996).
[25]B. Oakley II, "A virtual classroom approach to teaching circuit analysis," IEEE Trans. Educ. **39**, 287 (1996).
[26]E. C. Shaffer and F. J. Mabry, "A student designed, Web-based learning program for circuit analysis," in *Proc. 2000 Frontiers in Education Conf.* (2000), p. T2D.
[27]A. Luchetta, S. Manetti, and A. Reatti, "SAPWIN-A symbolic simulator as a support in electrical engineering education," IEEE Trans. Educ. **44**, CDROM (2001).
[28]L. Palma, R. F. Morrison, P. N. Enjeti, and J. W. Howze, "Use of Web-based materials to teach electric circuit theory," IEEE Trans. Educ. **48**, 729 (2005).
[29]B. P. Butz, M. Duarte, and S. M. Miller, "An intelligent tutoring system for circuit analysis," IEEE Trans. Educ. **49**, 216 (2006).
[30]B. P. Butz and S. M. Miller, Evaluation of IMITS for the National Science Foundation, http://www.temple.edu/imits/shock/Evaluation.pdf.
[31]L. Weyten, P. Rombouts, and J. De Maeyer, "Web-based trainer for electrical circuit analysis," IEEE Trans. Educ. **52**, 185 (2009).
[32]P. D. Cristea, R. I. Tuduce, and A. R. Tuduce, "Knowledge assessment in intelligent e-learning environments," in *Internat. Conf. on Signals & Electronic Systems* (Poznan, Poland, 2004), p. 573.
[33]P. Cristea and R. Tuduce, "Automatic generation of exercises for self-testing in adaptive e-learning systems: Exercises on AC circuits," in *3rd Workshop on Adaptive and Adaptable Educational Hypermedia at the AIED'05 conference (A3EH)*, edited by A. I. Cristea, R. Carro, and F. Garzotto (Amsterdam, 2005), p. 28. http://hcs.science.uva.nl/AIED2005/W9proc.pdf.
[34]E. G. Torres, T. Iida, and S. Watanabe, "Measuring the student knowledge state in concept learning: An approximate student model," IEICE Trans. Inf. & Syst. **E77-D**, 1170 (1994).
[35]S. Watanabe, J. Miyamichi, and I. R. Katz, "Teaching circuit analysis: A mixed-initiative intelligent tutoring system and its evaluation," in *Proc. IFIP TC3 Internat. Conf. on Advanced Research on Computers in Education (ARCE'90)*, edited by R. Lewis and S. Otsuki (Tokyo, 1991), p. 19.
[36]T. Grabowiecki, "Expert system for teaching of problem solving in circuit theory," in *Europ. Conf. on Circuit Theory & Design* (Stresa, Italy, 1999), p. 1283.
[37]D. L. Millard, "Interactive learning modules for electrical engineering education," in *Electron. Compon. and Technol. Conf.* (2000), p. 1042.
[38]D. Millard and G. Burnham, "Interactive educational materials and technologies," in *Internat. Conf. Engrg. Educat.* (Manchester, U.K., 2002), p. 445.
[39]D. Millard and G. Burnham, "Increasing interactivity in electrical engineering," in *33rd ASEE/IEEE Frontiers in Educat.* (IEEE, Boulder, CO, 2003), p. F3F8.
[40]A. Micarelli, F. Mungo, F. S. Nucci, and L. Aiello, "SAMPLE: An intelligent educational system for electrical circuits," J. Artific. Intell. Educat. **2**, 83 (1991).
[41]Z. C. Zacharia, "Comparing and combining real and virtual experimentation: An effort to enhance students' conceptual understanding of electric circuits," J. Computer Assisted Learning **23**, 120 (2007).
[42]B. Oakley II, *CircuitTutor* (Addison-Wesley, Reading, MA, 1992).
[43]A. M. Davis, *Linear Circuit Analysis* (PWS Publishing Co., Boston, 1998).
[44]A. Ioinovici, *Computer-Aided Analysis of Active Circuits* (Marcel Dekker, New York, 1990).
[45]R. W. Jensen and B. O. Watkins, *Network Analysis* (Prentice-Hall, Englewood Cliffs, NJ, 1974).
[46]J. Grimbleby, *Computer-Aided Analysis and Design of Electronic Networks* (Pitman, London, 1990).
[47]J. D. Irwin and R. M. Nelms, *Basic Engineering Circuit Analysis* (Wiley, Hoboken, NJ, 2010).
[48]W. H. Hayt Jr., J. E. Kemmerly, and S. M. Durbin, *Engineering Circuit Analysis* (McGraw-Hill, New York, 2011).
[49]C. K. Alexander and M. N. O. Sadiku, *Fundamentals of Electric Circuits* (McGraw-Hill, New York, 2008).
[50]R. C. Dorf and J. A. Svoboda, *Introduction to Electric Circuits* (Wiley, Hoboken, NJ, 2006).
[51]J. W. Nilsson and S. A. Riedel, *Electric Circuits* (Prentice-Hall, Boston, 2011).
[52]R. E. Thomas, A. J. Rosa, and G. J. Toussaint, *The Analysis and Design of Linear Circuits* (Wiley, Hoboken, NJ, 2009).