Automatic Circuit Analysis Problem and Solution Generation

James Macindoe[a] and Jonathan C. Li[a].
Department of Electrical and Computer Systems Engineering, Monash University[a]
Corresponding Author Email: jsmac5@student.monash.edu

Structured Abstract

BACKGROUND
The context of this work is introductory electrical engineering circuit analysis courses. A program has been developed which generates circuit analysis problems typical of such courses. It also generates detailed, error-free solutions using the techniques taught in these courses. This includes random topologies, and not just element values.

PURPOSE
The practice of generating questions with random element values is already common place and some work has also been done on generating random topologies (Whitlatch et al. 2012). The aim of this work is to produce a more comprehensive system which covers most of the techniques taught in a first course on circuit analysis. The problems will also be generated completely from scratch and include solutions. This work could then be integrated into a Computer-Aided Instruction (CAI) system for students. It could also be used by instructors to generate exam or tutorial questions or could be included in a quiz system where students are presented with questions with different topologies but similar difficulties to discourage copying answers.

DESIGN/METHOD
The system has been developed with reference to the course material used at Monash University, and particularly with reference to introductory circuit analysis textbooks. The aim has been to generate questions like those found in textbooks. Importantly this includes generating detailed solutions which can step a student through the problems in the same manner as textbook examples.

RESULTS
The system is capable of generating a range of problems. It supports nodal analysis, mesh analysis, resistor simplification, ohm's law, Kirchhoff's voltage law (KVL), Kirchhoff's current law (KCL), voltage division and current division. It can provide the user with an endless supply of new and unique problems which cover any of the above techniques they wish to study or test. This could be of use as an alternative to textbook questions, and could be integrated into an online system where additional features could be implemented, such as picking up student errors as soon as they are made instead of only when they check the solution.

CONCLUSIONS
New algorithms are presented which generate circuit analysis problems and find solutions to those problems. The problems and solutions are typical of introductory circuit analysis textbooks and cover a range of the circuit analysis techniques taught in introductory courses. These algorithms can be used to generate problems that meet the exact needs of a particular student or test.

KEYWORDS
Computer-aided instruction, linear circuit analysis

Introduction

Online learning is being used increasingly in higher education. It can be particularly effective when combined with classroom instruction (Sitzmann, Kraiger, Stewart, & Wisher, 2006). One application of web technologies is to provide online assessment, such as a quiz system. Some systems have been built which provide advanced features based on knowledge of the domain being taught (e.g. Vanlehn et al., 2005). This can potentially provide a number of advantages such as giving immediate feedback to students, including on individual solution steps instead of just answers, providing hints on which technique to apply next and adapting to individual student needs, for example by generating individualised questions (Woolf 2010).

Despite these advantages, the software used for e-learning in circuit analysis remains fairly primitive. Generic quiz software can of course be used, such as Moodle's quiz module. It is common to randomly generate component values in such quizzes to discourage the copying of answers. It is also common for textbook publishers to provide accompanying websites. However, relatively little work has been done on generating entirely new questions, including random circuit topologies (Whitlatch, Wang, & Skromme, 2012).

An algorithm for generating random circuits is outlined by Whitlatch et al. (2012) and evaluated in (Skromme, Wang, Reyes, Quick, Atkinson and Frank, 2013) and (Skromme, Wang, Reyes, Quick, Atkinson and Frank, 2013). This paper extends this work to generate a wider range of problems. A number of performance improvements over the generation algorithm given by Whitlatch et al. (2012) are also presented. However, the main contribution is in a new solution generation engine. Whitlatch et al. (2012) only cover the simple cases of nodal and mesh analysis and resistor combination. This paper presents a more general solution generation engine which supports a broader range of problem types and analysis techniques. It supports testing specific techniques and ideas (e.g. testing just Kirchhoff's current law (KCL) instead of all the concepts required for nodal analysis) and it is easy to add support for new techniques to the engine. At present support has been added for nodal analysis, mesh analysis, resistor simplification, ohm's law, Kirchhoff's voltage law (KVL), Kirchhoff's current law (KCL), voltage division and current division. Support for additional techniques will be added in the near future.

Hopefully, the algorithms this paper describes can ultimately be included in existing software systems, such as the websites provided by textbook publishers. The authors also plan to implement a web interface for the system to demonstrate its potential.

Characterisation of circuit analysis problems

A large number of elementary circuit analysis problems were examined to characterise their typical properties (mainly from (Irwin and Nelms, 2008), (Alexander and Sadiku, 2007) and (Nilsson and Riedel, 2011)). Fortunately most problems have enough in common to make automatic generation feasible. Almost all problems include a circuit to be analysed (some do not, such as short answer questions, but they are not considered here). Typically the student will be asked to find one or more quantities in the circuit (voltages, currents, resistances, etc). Although most of the problem statements appear to follow a similar formula, the solutions can often be vastly different from each other due to the wide range of circuit analysis techniques that are available.

A single circuit generation module and two solution generation modules have been developed to deal with this variation. The variation in the circuits is small enough to be captured in a single algorithm for their generation. After examining a large number of question solutions, it was observed that questions are generally solved in one of two ways:

1. Using nodal or mesh analysis to generate a set of equations and then solving these equations simultaneously.

2. Applying a series of circuit analysis techniques, one after the other, to progressively solve the circuit one quantity at a time.

These two types of solutions correspond to our two solution generation modules. The first type of question is relatively easy to implement a solver for, since it is algorithmic. The authors are not aware of any other work which presents a solution generation engine for the second type of question.

Example output

The program produces a wide range of problems. One example is shown in Figure 1. Work on graphical support in a web interface is underway.

```
== Question ==
Find iv0 given r0 = 5.0 kOhm, r = 1.0 kOhm, i0 = 100.0 mA,
r1 = 4.0 kOhm

                 ---i2-->
     0---------0----r1---1
     |   ^     |   |     |   |
     iv0 i0    r0  i1    r2  i3
     |   |     |   v     |   v
     g---------g---------g


== Solution ==
Transform to the following circuit

                 ---i2-->
     0---------0----r3---g
     |   ^     |   |     |
     iv0 i0    r0  i1    |
     |   |     |   v     |
     g---------g---------g


Combining series resistors, r3 == r1 + r2, we find r3
Using current division, i1 == i0*r3/(r0 + r3), we find i1
Using ohm's law, v0 == i1*r0, we find v0
From the voltage supply, v0 == iv0, we find iv0
```

Figure 1: Text representation of output. 'g' is the ground node.

Circuit Generation

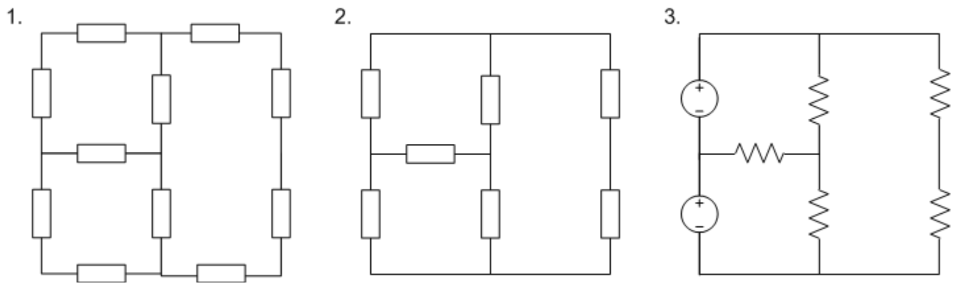

Figure 2: Circuit generation algorithm: 1. choose topology 2. add shorts 3. place circuit elements

Our circuit generation algorithm is a modified version of the algorithm presented by Whitlatch et al. (2012). It is illustrated in Figure 2. In the first step a 'topology' is chosen, meaning a circuit layout where the position of 'opens' is already determined. The position of 'shorts' and circuit elements has not yet been determined, so these are shown as generic boxes. In the second step, the position of shorts is determined, with the remaining positions to be turned into circuit elements. In the third step, specific circuit elements are chosen and placed. During each step, it is easy to generate an invalid circuit. The criteria for a valid circuit are described below and then the methods used to avoid such circuits during each step are outlined.

Data structures

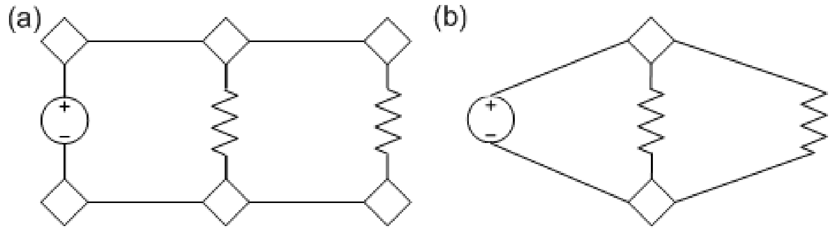

Figure 3: (a) Net and (b) Grid data structures. Diamonds represent graph nodes. The graph edges are the circuit elements and, for the grid, the shorts.

There are two data structures used during the generation process, which are referred to as the ‘grid’ and the ‘net’. They are shown in Figure 3. The grid is the layout of the circuit as it would be printed on paper, while the net represents only its electrical properties, much like a SPICE netlist. Both data structures are graphs. The ‘python’ programming language’s NetworkX library was used, which also allows data to be stored on any node or edge.

The grid has nodes at all integer coordinates on an $n \times m$ Cartesian plane. Edges can exist connecting a node to its immediate neighbours (up, down, left or right). The edges connecting the nodes can represent circuit elements or shorts (opens are represented by there being no edge between those nodes). The particular element on the edge is stored in the edge’s data structure.

The net represents the electrical structure of the circuit. Its nodes represent electrical nodes. The edges represent elements between electrical nodes. Shorts are not represented, since they form part of the electrical nodes.

Both data structures are generated simultaneously, such that at all times there is a correspondence between the two structures. Another approach would have been to generate the net first, and then try to find an appropriate layout (grid), but finding good layouts for graphs is a well known difficult algorithmic problem. Another would be to generate the grid first and then extract the net. This is the approach taken by Whitlatch et al. (2012) but it makes it difficult to evaluate the electrical properties of the circuit during generation. By taking these into account, a number of the problems in the algorithm presented by Whitlatch et al. (2012) are avoided.

Criteria for valid circuits

It is possible to generate invalid questions which have no solution, as well as questions with undesirable qualities. This has been described extensively by Whitlatch et al. (2012). In brief, situations where the nodal or mesh equations are not solvable must be avoided, such as when sources are arranged inconsistently. Furthermore, there are certain circuit configurations which, though valid, are undesirable. For example, if a circuit element is completely shorted or the circuit is hinged. The way in which these circuits are avoided is described below for each step.

Circuit specification input

The circuit generation algorithm takes as input the number of each type of circuit element to include and optionally also takes a specification of the number of meshes and/or the number of nodes. Circuits are constrained by the well known equation $N = B - M + 1$ where N is the number of nodes, B is the number of branches and M is the number of meshes. B can be calculated from the list of elements to be included in the circuit. If either N or M are not specified, any values can be chosen which satisfy the above equation.

Step 1: Choose circuit topology

In this step, a circuit topology is chosen in which the positions of opens are already determined. The positions of shorts and circuit elements are yet to be determined and they are left as 'generic' elements. This step is shown in Figure 2. It is similar to step 1 in (Whitlatch et al. 2012). However, they being with an empty grid and then randomly place wire segments, using a heuristic to try to guide the process towards generating a valid topology. If an invalid topology is generated, or the topology had the wrong number of meshes, the process is restarted.

Although the number of possible circuits a student could be given is very large, and thus requires a generation process, it was observed that the number of different topologies used in these circuits is nonetheless quite small. This is because the circuits are generally quite small (rarely larger than a 3x3 grid). So instead, all the topologies are pre-generated and then one that has the specified number of meshes is randomly chosen. This is done by beginning with the set of $n \times m$ grids ($1 \leq n, m \leq 4$) without any opens and then deriving new topologies by iterating through different ways opens could be added to these grids. The runtime of this code is very modest (imperceptible on a consumer grade computer) and only has to be run once, after which its results can be reused.

Step 2: Add shorts

Step 1 determined the number of meshes in the circuit. This step determines the number of branches. After selecting a topology with the right number of meshes, its edges are randomly iterated through, converting them into shorts, until the number of branches in the circuit is as specified. There are two ways to generate an invalid circuit which must be avoided:

(a) causing an element to short circuit, since shorted elements serve no purpose, and

(b) 'hinging' the circuit, since this will leave the circuit with two unrelated components.

Whitlatch et al. (2012) restart the process if either of these events occur. In this work, this is avoided using the net data structure. The only way for an element to become shorted is if there are two parallel elements both connecting the same two electrical (net) nodes, and one is then converted to a short, causing the remaining element to be short circuited. This can be avoided in O(1) time by checking for this situation in the net before converting an element to a short.

The circuit is hinged when there is a path of shorts through the interior of the circuit which connects two points on the exterior. This can also be avoided in O(1) time by checking if an element that is about to be converted to a short is in the interior and connects two electrical (net) nodes which are on the exterior. Each net node and element is labelled as being on the interior or exterior when the topology is generated and then this information is updated as shorts are added.

Step 3: Place circuit elements

In the final step, the generic elements are replaced with specific elements (resistors, capacitors, etc). As described by Whitlatch et al. (2012) and Wadhwa (2007), this must be done using a minimum spanning tree (MST) of the net data structure. The branches in the MST are referred to as twigs, while the other branches are called links. Voltage sources must be placed on the twigs and current sources on the links. For an AC question, resistors, capacitors and inductors can then be placed randomly anywhere. For a DC question, resistors can be placed anywhere, but inductors must be placed on twigs and capacitors on links.

Solution Generation

Two solution generation engines have been developed for the two types of problems described in the ‘Characterisation of circuit analysis problems’ section. The first is for nodal and mesh analysis and follows the algorithms as described in most textbooks (e.g. Irwin and Nelms, 2008), so is not described here. The second module solves most other problems and is based on graph theory.

Although nodal and mesh analysis are very general and easy to implement, they do not allow for questions and solutions that test the full range of techniques taught to students. Furthermore, it would be better to present detailed step-by-step solutions and not just a numerical answer. Many questions are typically solved by the application of a series of network laws and network transformations, one after the other, to solve the problem one quantity at a time. It can often seem as if an intuition is required to know when it is appropriate to apply different circuit analysis techniques. This has been investigated by Sussman and Stallman (1975) and Stallman and Sussman (1977) and they found it was possible for a computer to apply these types of techniques.

Our engine represents the network equations and electrical quantities in a graph and then emulates student problem solving strategies by running an algorithm on the graph. The result is a solution like those presented in textbook examples.

Equation Graph

Graphs similar to the ones used here have been described by De Kleer and Sussman (1980). Their graphs have a node for each quantity and each equation, with edges between each equation and any quantities in that equation. An example is shown in Figure 4. Our graph is the same, but it is more convenient to think of it as a hypergraph with the quantities as the nodes and the equations as the edges. That terminology will be used here.

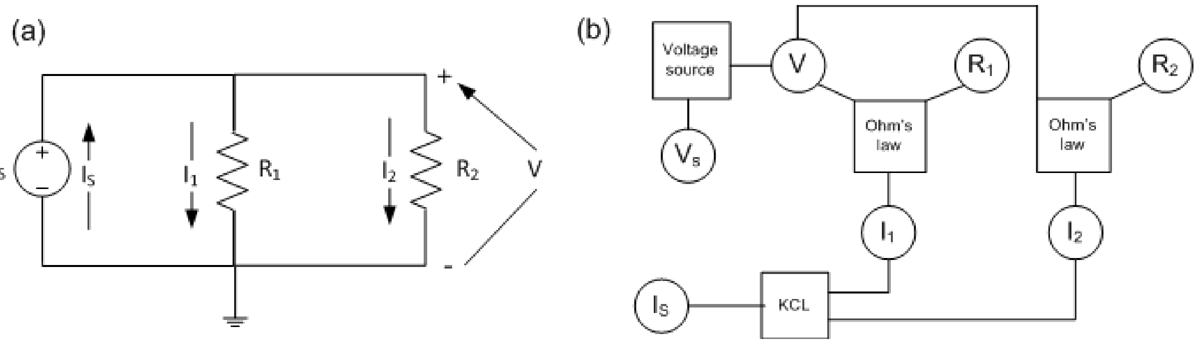

Figure 4: (a) example circuit (b) corresponding equation graph. (De Kleer and Sussman 1980).

Algorithm

Initially, we know the values of the quantities given in the question. An equation can be solved if every quantity in it except for one is known. That quantity can then be solved for. Thus, if every quantity connected to an edge (equation) but one is known, that edge can be traversed to find the remaining quantity. This may in turn allow equations connected to that quantity to be solved. Often, this process can be continued until every quantity has been found. Otherwise, simultaneous equations must be used to solve the problem, which this solution engine does not consider.

Thinking of the problem this way effectively reduces it to the well known shortest path problem. The path represents the progressive steps taken by the student. Passing through an equation represents using it and reaching a quantity represents having solved for that quantity. Dijkstra’s algorithm is one solution to the shortest path problem and is shown in Figures 5 and 6. However, a few modifications for our problem must be made. Initially, every given quantity will be initialized to have a path cost of zero. Furthermore, on lines 7 and 8 of Figure 5c, only edges that can actually be traversed (only have one unknown) are relaxed.

```
(a) INITIALIZE-SINGLE-SOURCE($G, s$)
1  for each vertex $v \in G.V$
2      $v.d = \infty$
3      $v.\pi = $ NIL
4  $s.d = 0$

(b) RELAX($u, v, w$)
1  if $v.d > u.d + w(u, v)$
2      $v.d = u.d + w(u, v)$
3      $v.\pi = u$

(c) DIJKSTRA($G, w, s$)
1  INITIALIZE-SINGLE-SOURCE($G, s$)
2  $S = \emptyset$
3  $Q = G.V$
4  while $Q \neq \emptyset$
5      $u = $ EXTRACT-MIN($Q$)
6      $S = S \cup \{u\}$
7      for each vertex $v \in G.Adj[u]$
8          RELAX($u, v, w$)
```

Figure 5: Dijkstra's algorithm. From (Cormen et al., 2001). $v.d$ is an estimate (upper bound) on the shortest distance to v, $v.\pi$ is v's predecessor, w is the edge cost, S is the set of finalised nodes and Q is a min-priority queue, sorted by d.

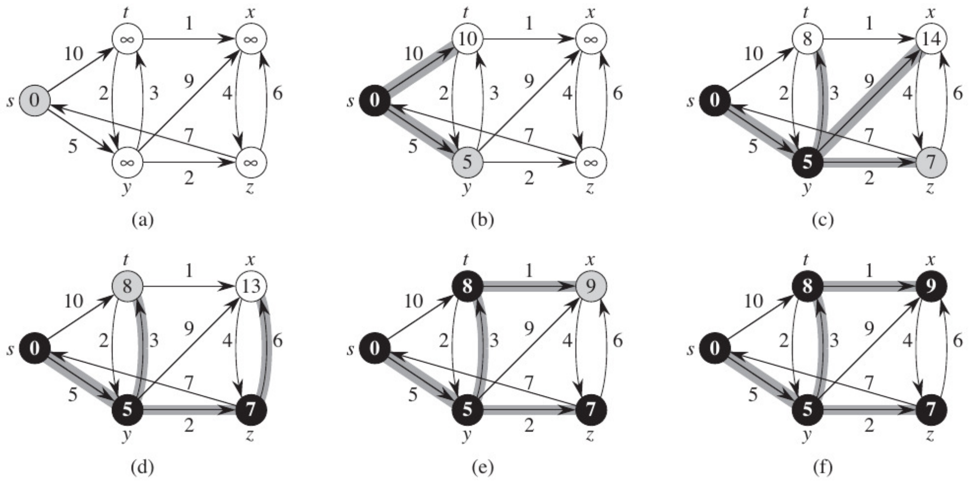

Figure 6: Illustration of Dijkstra's algorithm. From (Cormen et al., 2001).

Path cost

The choice of path cost in Dijkstra's algorithm is important because the solution with the lowest path cost will be the output suggested solution. Scores were assigned to each circuit analysis technique based on how difficult they were judged to be. For example, Ohm's law was judged to be 'easy' and so was given the low score of 0.25. KVL, on the other hand, was deemed more difficult and so was given the higher score of 1.0. The path cost is then simply the sum of the scores of each edge (equation) passed through to get to a node. The choice of score values could be improved by extracting them from empirical data. For example, once the system is deployed, it would be possible to examine which questions students tend to have difficulty with.

It is easy to implement this path cost scheme for equations but is slightly more difficult for network transformations. A transformed circuit will have a number of new equations. The cost of the transform should be added the first time one of these equations is traversed, but should not be added again when another equation from the circuit is traversed. To implement this, the algorithm tracks which transformations have been used to solve each quantity. The cost of an edge is then the equation's cost plus the cost of any transformation used to find any of the quantities attached to the equation plus the cost of any additional transformations needed to make the equation available, making sure to count the cost of each transformation only once.

Ordering solution steps

Dijkstra’s algorithm outputs the set of nodes on the shortest path to each quantity. However, our path ‘branches’ at each equation it passes through, since every quantity in that equation had to be solved. It is therefore more difficult to choose a logical order to print the steps. The solution was to record the order in which the algorithm solved each quantity and then print them in that same order.

Supported Techniques

So far, support for a number of different analysis techniques has been added. Mathematical laws can be implemented simply by writing code to extract the relevant equations from the net data structure. So far, support has been implemented for ohm's law, KCL, KVL, voltage division and current division. Network transformations require new circuits to be generated from the old ones. So far, support has only been implemented for series and parallel resistor transformations. In future, support will be added for source transformations and Thevenin and Norton's Theorems. Support could also be added for superposition, which should be a simple matter of applying the algorithm to each of the simpler superposition circuits and then combining the solutions. Ideally, support for op-amps and transient circuits will also be added. However, these may require additional data structures and solution engines.

Suggested Applications

A number of possible uses are envisaged for the algorithms described here. Namely, as a backend for an online quiz system, as a tool to aid writing exams and tutorials and as a self-study tool for students. Work on a web interface which would quiz students is underway. Ideally, every student could be given a completely unique set of questions, so as to prohibit copying answers from classmates or textbook solutions manuals. However, the questions would have to be of equal difficulty. This could be done using the path cost function described for the solution generator.

At present, the system takes a list of circuit elements as its input. The program generates a circuit meeting those specifications and then trials different allocations of quantities to be provided in the question. For each allocation, it runs the solution engine. It then outputs the ‘best’ allocation (at present, the one that results in the hardest question). A potentially better approach would be for the user to specify a set of techniques to be tested and a question difficulty (in terms of the path cost). This could be done by searching for an allocation of given quantities that results in the specified question and restarting the entire process if no suitable allocation can be found. Alternatively, if the algorithms are implemented in server-side software, it could be run over a long period of time to generate many questions which could be stored. When a user needs a question meeting particular specifications, this question could be chosen from the pre-generated store.

It is also worth considering if these techniques could be applied to other areas of engineering. It is expected that they could be applied to more advanced courses in electrical engineering which follow similar patterns of circuit analysis, only with more advanced techniques. It may be possible to apply them to other areas of engineering. However, they work particularly well here because circuit analysis questions all have a lot in common. Namely, they all begin with a circuit and are solved by applying a limited number of techniques. Often in other fields, the types of questions set are much more varied and this makes automatic generation difficult.

Conclusion

Algorithms for the generation of circuit analysis problems and automated finding of solutions have been described. An improved algorithm for circuit generation has been presented, as well as a new solution engine which allows for a much wider range of problems to be generated. This includes support for nodal analysis, mesh analysis, resistor simplification, ohm's law, Kirchhoff's voltage law (KVL), Kirchhoff's current law (KCL), voltage division and current division. It should be possible to add support for additional techniques and support will be added in future for source transformations, superposition and Thevenin and Norton's Theorems. These algorithms could be applied in many different systems, such as quiz systems and student study aids. Work on a web based quiz system which leverages the current backend is underway.

References

Alexander, C. K. & Sadiku, M. N. O., (2009). *Fundamentals of electric circuits.* McGraw-Hill Higher Education.

Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2001). *Introduction to algorithms*. Cambridge: MIT press.

De Kleer, J., & Sussman, G. J. (1980). Propagation of constraints applied to circuit synthesis. *International Journal of Circuit Theory and Applications*, 8(2), 127-144.

Irwin, J. D., & Nelms, R. M. (2008). *Basic engineering circuit analysis*. Wiley Publishing.

Nilsson, J. W., & Riedel, S. (2011). *Electric Circuits*. Pearson Education.

Sitzmann, T., Kraiger, K., Stewart, D., & Wisher, R. (2006). The comparative effectiveness of web-based and classroom instruction: A meta-analysis. *Personnel Psychology*, 59(3), 623-664.

Skromme, B. J., Rayes, P. J., Whitlatch, C. D., Wang, Q., Barrus, A., Quick, J. M., ... & Frank, T. S. (2013, October). Computer-aided instruction for introductory linear circuit analysis. In *Frontiers in Education Conference, 2013 IEEE* (pp. 314-319). IEEE.

Skromme, B. J., Wang, M. Q., Reyes, P., Quick, J. M., Atkinson, R. K., & Frank, T. (2013). Teaching linear circuit analysis techniques with computers. In *Proceedings of the 2013 American Society for Engineering Education Annual Conference & Exposition.*

Stallman, R. M., & Sussman, G. J. (1977). Forward reasoning and dependency-directed backtracking in a system for computer-aided circuit analysis. *Artificial intelligence*, 9(2), 135-196.

Sussman, G. J., & Stallman, R. (1975). Heuristic techniques in computer-aided circuit analysis. *Circuits and Systems, IEEE Transactions on*, 22(11), 857-865.

Vanlehn, K., Lynch, C., Schulze, K., Shapiro, J. A., Shelby, R., Taylor, L., ... & Wintersgill, M. (2005). The Andes physics tutoring system: Lessons learned. *International Journal of Artificial Intelligence in Education*, 15(3), 147-204.

Wadhwa, C. L. (2007). *Network Analysis & Synthesis (Including Linear System Analysis*). New Age International.

Whitlatch, C. D., Wang, Q., & Skromme, B. J. (2012). Automated problem and solution generation software for computer-aided instruction in elementary linear circuit analysis. In *Proceedings of the 2012 American Society for Engineering Education Annual Conference & Exposition.*

Woolf, B. P. (2010). *Building intelligent interactive tutors: Student-centered strategies for revolutionizing e-learning*. Morgan Kaufmann.

Copyright statement