

# **NP-Completeness for All Computer Science Undergraduates: A Novel Project- Based Curriculum**

*Andrea F. Lobo and Ganesh R. Baliga*  
*Department of Computer Science*  
*Rowan University*  
*210 Mullica Hill Road, Glassboro, NJ 08028*  
*(856) 256- 4500 x3815, x3890*  
*lobo,baliga@rowan.edu*

## **ABSTRACT**

It is widely recognized that there are some aspects of intractability and computational complexity that every CS professional should understand and be able to apply. NP-completeness is one of these fundamental concepts. This paper describes a novel project-based curriculum for teaching NP-completeness, classic algorithm design techniques, and techniques for solving intractable problems. The proposed set of projects aims to help all Computer Science undergraduate students learn more about basic computability and NP-completeness, apply the scientific method, apply classic algorithm design techniques, implement algorithms for intractable problems using more advanced techniques, gain exposure to state of the art research, implement sophisticated algorithms and data structures, and apply algorithm design techniques that they have understood to a new domain. The curriculum also seeks to motivate a few undergraduates to pursue research. The set of projects is based on the well-known, NP-complete 3-Satisfiability problem (3SAT). We have used these projects in three offerings of our undergraduate Algorithm Design and Analysis course, while continuing to address the traditional course content. The informal assessment conducted during these course offerings indicates that typical undergraduate students achieve the targeted learning outcomes after completing the proposed curriculum.

## 1 INTRODUCTION

Intractability and computational complexity are fundamental Computer Science (CS) concepts. They are theoretically important and certainly of interest to academics. Additionally, there are aspects of intractability and computational complexity that every CS professional should understand and be able to apply: How to evaluate the efficiency of an algorithm for a problem; the fact that some problems are “hard” or “intractable” and cannot be solved efficiently in polynomial time, no matter how fast the processor or large the memory; the fact that there are some other problems, which are called NP-complete, whose exact complexity is not even known. It is widely recognized that these topics have an important place in the CS undergraduate curriculum. The ACM-IEEE CC2001 core [25] includes 6 hours of basic computability in topic AL5. Topic AL6 of CC2001 is “The Complexity Classes P and NP.” The ABET accreditation standards for CS programs [1] also require theory.

The NP-complete problems have practical as well as theoretical importance. Many NP-complete problems arise from natural applications, such as “Given a map with cities and roads with travel times, can a salesperson visit each city only once and return to the starting point given a number of days for the trip?” Another example of a NP-complete problem is “Given geographical locations for several cellular towers, their transmission powers, and a set of available transmission frequencies, can each tower be assigned a frequency so that overlapping coverage areas use different frequencies?” The well-known NP-complete Satisfiability problem (SAT) asks whether the boolean variables of a given formula in Conjunctive Normal Form can be assigned values that make the formula true [4]. While it may seem at first that SAT is a theoretical musing, there are lucrative commercial applications of SAT in circuit verification [12]. As the prices and capabilities of computers have continued to improve in recent years, there has been increased interest in algorithm design techniques for creating practical solutions to intractable problems. These techniques have applications beyond SAT and circuit verification, in burgeoning areas such as bioinformatics and cryptography [26,15]. Thus, it is becoming more important for CS undergraduates to understand them, even for those students who intend to join the workforce upon graduation.

The common curricular approach to computability in undergraduate CS programs is to address intractability and NP-completeness in an Algorithm Design and Analysis (AD&A) course. We have observed, anecdotally and from our experience with the accreditation of CS programs, that many AD&A courses and textbooks [3,4,6,8,16,17,21,27,30,31] address this topic towards the end. In many AD&A courses where this topic is discussed, the students are asked to write a simple NP-completeness proof. This is primarily an application of non-trivial mathematical skills, and many students do not develop an

understanding for the fundamentally difficult nature of NP-complete problems or the importance of the  $P = NP$  question. Additionally, many AD&A courses do not discuss or apply the algorithm design techniques used to solve intractable problems. These shortcomings of the common curricular approach to computability have not been adequately addressed in the Computer Science Education literature; searches of the ACM Digital Library and the reports of funded CCLI proposals result in a handful of moderately relevant hits [19].

This paper describes a project-based curriculum for teaching NP-completeness, classic algorithm design techniques, and techniques for solving intractable problems. We have used this novel approach and several combinations of the projects in three offerings of our undergraduate AD&A course since 2003. The informal assessment conducted during these course offerings indicates that average undergraduate students are capable of understanding and applying these concepts, and that a small amount of class time is needed to implement the new approach. We believe the wider adoption of these ideas would improve the preparation of undergraduate CS students for applied employment upon graduation, and their motivation and preparation for graduate school and further scientific pursuits.

## **2 A PROJECT-BASED CURRICULUM: GOALS, LEARNING OUTCOMES AND MOTIVATION**

The goals of the project-based curriculum described in this paper are to help undergraduate CS students understand NP-completeness, apply classic algorithm design techniques, and apply techniques for solving intractable problems. These projects are learning tools for important CS concepts and the scientific method. The projects aim to help all CS undergraduate students achieve the following targeted learning outcomes:

1. Learn more about basic computability and NP-completeness;
2. Apply the scientific method, as they incrementally develop an efficient programmatic solution to a difficult problem;
3. Apply classic algorithm design techniques, such as exhaustive search, backtracking, and greedy heuristics;
4. Implement algorithms for intractable problems using more advanced techniques, such as hill-climbing and randomization;
5. Gain exposure to state of the art research;
6. Implement sophisticated algorithms and data structures; and
7. Apply algorithm design techniques that they have used and understand to a new domain.

The eighth objective pertains to a small subset of students: To motivate a few undergraduates to pursue research.

We use a set of programming projects based on the Satisfiability (SAT) problem to help the students in our undergraduate Algorithm Design and Analysis (AD&A) course achieve the learning outcomes of the preceding paragraph. Why project-based learning? Why SAT? Why the AD&A course? The remainder of this section motivates these choices.

Project-Based Learning is well accepted in the CS education community as an effective teaching/learning tool. We posit that expanding the exposure of the typical CS undergraduate to NP-completeness beyond abstract definitions and proofs leads to a deeper understanding and appreciation of the concept. A single programming assignment to implement a solution to an NP-complete problem is not likely to give the typical undergraduate a good feel for the importance of the topic. One project to implement a simple brute force algorithm and execute it on small and medium size inputs illustrates an exponential complexity but not the inherent difficult nature of the problem. Similarly, one project to implement a more efficient algorithm with sophisticated data structures can seem like a challenging academic exercise but may not provide the students with insights or opportunities for discovery. We propose using a set of programming projects to help students achieve the stated learning outcomes in a more concrete manner. Our earlier work on two NSF-funded projects [citations omitted] demonstrated that the use of real-world objects in CS1 Labs significantly enhances student interest in, and comprehension of, the object oriented approach. Similarly, in the case of NP-completeness, using the hands-on programming projects is likely to result in increased student interest and comprehension, compared to a more abstract presentation of the topic. The informal assessment results from our experiences using the set of NP-complete projects suggest that this indeed the case.

The proposed set of projects is based on the well-known 3-Satisfiability problem (3SAT). The classic NP-complete problem of 3SAT [13] is defined as follows: Given a propositional logic formula in conjunctive normal form where each clause has exactly three Boolean variables, is there an assignment of true/false values to the variables in the formula that makes the formula evaluate to true? From a purely theoretical point of view, all NP-complete problems are reducible to one another, and therefore, the choice of problem makes no difference. 3SAT is a particularly good application domain for the proposed set of projects for numerous reasons. Firstly, SAT is a classic NP-complete problem: The seminal paper on NP-completeness by Stephen Cook [32] proved that it is NP-complete. Secondly, 3SAT is relevant today, since the published research literature on 3SAT has been especially active during the past decade. Thirdly, lucrative applications of 3SAT exist: Efficient 3SAT solvers have recently been used in applications ranging from digital circuit design

to theorem proving [12, 20]. Additionally, there is an active annual international competition of efficient SAT solvers that undergraduate students can participate in [2], and public domain repositories of benchmark solvers and “hard” 3SAT instances are available. Finally, the definition of 3SAT is so simple that a professor can present it for a class of CS sophomores in less than five minutes.

The set of programming projects based on 3SAT is designed for use in undergraduate Computer Science (CS) programs. The projects would typically be used in an Algorithm Design and Analysis (AD&A) course, but could also be used in courses on Theory or Data Structures. A professor teaching the AD&A course could use all or some of the projects in the set, possibly along with other programming projects on traditional AD&A topics of the professor’s choice. We have used the 3SAT project-based curriculum in our undergraduate AD&A course to help students achieve the targeted learning outcomes. On the first class of the semester, we define 3SAT and assign the implementation of an exhaustive-search, brute-force solver. The students work on the remaining projects of the set throughout the semester. Students are more motivated to understand any class discussions or reading assignments on 3SAT when it is the subject of a concrete, current assignment. Short discussions on NP-completeness and the projects are interspersed throughout the semester, as the course continues to cover the usual topics. Each project adds a new aspect, and the students progressively gain a deeper understanding of NP-completeness during the entire semester. Our preliminary assessment of this approach indicates that the resulting student experience is more meaningful than “covering” NP-completeness in the final weeks of the semester.

### **3 THE SET OF PROJECTS**

We have designed a curriculum consisting of a set of programming projects based on 3SAT to help students achieve the targeted learning outcomes. The first five projects ask the students to implement and evaluate a 3SAT solver. The details of each project are presented later in this section. We also describe the contributions of the projects to the learning outcomes. Let us first outline the basic structure and content of the set of projects:

1. Brute force solver: Problem definition, exhaustive search
2. Systematic search solver: Backtracking (recursive and non-recursive), trees, traversals
3. Enhancements to the systematic search solver: Greedy design techniques, heuristics, sophisticated data structures
4. Comparison of probabilistic search strategies: Hill-climbing, randomization, local search, iterative repair, simulated annealing

5. Back jumping: Graph search
6. Apply techniques to another NP-complete problem.

The reader will have noticed that much of this content is traditional in the AD&A course. The projects allow the professor to present topics such as backtracking, greedy heuristics or randomization in the context of a problem that the students are already familiar with, since they learned about 3SAT in the first programming project. The professor will probably want to provide other examples beyond 3SAT, but the projects provide a common thread for all these topics. The students are more likely to be interested since these topics are immediately relevant to their programming assignments. Our experience with the proposed set of projects, over three course offerings since 2003, suggests that an AD&A professor has time for discussions on NP-completeness and different 3SAT algorithms throughout the semester, while presenting the usual course content.

For project 1, the professor defines 3SAT on the first class of the semester. The students are able to understand the problem after a very brief explanation, building only on their prior knowledge of propositional logic. The professor immediately assigns the implementation of an exhaustive search solver that generates all possible true/false combinations for variables in the input formula and checks if any of these combinations makes the formula evaluate to true. Relying only on their knowledge of rudimentary data structures, the students can implement the brute force 3SAT solver in the first week of the semester. The students spend the next week creating the infrastructure to solve many problem instances and measure execution times. They test their programs using small 3SAT problem instances. Interesting 3SAT problem instances can be found in the online archives of the annual SAT competition [2]. The students collect data on their program execution times for increasing problem (formula) size, analyze the results, and obtain first-hand experience with the limitations of the exponential complexity.

Project 2 asks the students to implement the well-known Davis-Logemann-Loveland (DLL) [10] algorithm. Project 2 requires that the students understand and implement backtracking and tree traversals. Project 3 enhances the basic DLL algorithm by using a combination of greedy heuristics for branch variable selection [9] and fast data structures for unit propagation [18,22], all of which come from the recent research literature. The students implement project 3 and analyze its performance, which is vastly better than that of project 2. They “discover” that even the most cutting-edge algorithms which possibly search the entire search space are unable to process medium-sized 3SAT problem instances in a reasonable amount of time. We use this discovery to motivate the need for *incomplete solvers*, i.e., algorithms that do not always find an assignment of true/false values to variables in a satisfiable formula. The research area of incomplete solvers for the 3SAT problem has flourished in the past ten

years [5,28,29]. Incomplete solvers employ numerous advanced techniques that have applications beyond 3SAT, including local search, iterative repair, simulated annealing and randomization. Project 4 asks the students to compare the performance of two incomplete solvers, namely GSAT [29] and WALKSAT [28]. GSAT uses a hill climbing (greedy) approach whereas WALKSAT, which performs well in practice, mixes hill climbing with a random walk. Many variations of projects 3 and 4 are possible. A professor could require the implementation of other variants as part of the project sequence, or suggested them as extra-credit projects, or allow each student to pick his/her own variant from the literature, or assign different variants to different teams of students and have each team present its results to the class.

The first targeted learning outcome is to learn more about basic computability and NP-completeness. As the students work on the proposed projects throughout the semester, they gain hands-on, progressively more sophisticated experience with intractability. They also develop a good understanding of the 3SAT problem. When the time comes to actually cover NP-completeness, the professor can use 3SAT to motivate the topic. It should be easier for the professor to keep the attention of motivated students through the proofs and discussions. NP-completeness is not esoteric to these students, but very relevant, and the first targeted student learning outcome on learning about computability and NP-completeness is facilitated.

The second targeted student outcome is to apply the scientific method to incrementally develop a programmatic solution to a difficult problem. As stated in CC2001 [25, p46], “the process of abstraction (data collection, hypothesis formation and testing, experimentation, analysis) represents a vital component” of CS. The proposed projects ask the students to apply the scientific method throughout the semester. Our earlier description of Project 1 includes the student's first experience with data collection, experimentation and analysis. Once the students have analyzed and understood the weaknesses of the naïve brute force enumeration algorithm, they are introduced to backtracking. The students then hypothesize that backtracking may be the basis for a better solution, test this hypothesis, and analyze the results. Subsequent projects attempt to make further improvements on the performance of the SAT solver. The iterative process of solution design, implementation, testing, analysis, and redesign gives students experience applying the scientific method, the second targeted student outcome. In Project 4, for example, students “discover” that a greedy approach does not work for all problems instances or all problems. While the students are enjoying the puzzle of how to write a better 3SAT solver, they are retracing the footsteps of their ancestor researchers who previously traveled from simple brute force to more sophisticated approaches. The students are also experiencing the benefits of research, published results, and scientific advancement.

The third student learning outcome is to apply classic algorithm design techniques. The students will need to understand and implement classic algorithms and data structures to complete the projects, including trees, lists, graphs, exhaustive search, backtracking, hill climbing, randomization, and greedy algorithms.

The fourth student learning outcome is to design algorithms for intractable problems using advanced techniques. The design of algorithms using advanced techniques is clearly an objective of any AD&A course, but why *these* techniques? The proposed projects help students learn algorithm design techniques for intractable problems, such as randomization [23]. In the typical AD&A course, randomization comes up during the discussions on pivot selection for the Quicksort algorithm and hashing. We believe that the theoretical importance of this advanced algorithm design technique and its numerous applications in domains such as bio-informatics [7,26], computational geometry [24] and cryptography [15] justify its exposition at the undergraduate level. Our positive experiences using some 3SAT-based projects during three offerings of the AD&A course suggest that such advanced topics can be covered within the project framework without significant sacrifices in the traditional course content. The proposed project sequence allows students to apply these techniques to the 3SAT problem, after the students become familiar with the problem. In this way, the students deal with only one new issue at a time: a new problem and a known approach in Project 1, or a known problem and new approaches in the subsequent projects.

The fifth and sixth targeted student outcomes are to gain exposure to state of the art research and implement sophisticated algorithms and data structures. Projects 3 and 4 ask the students to implement sophisticated algorithms and data structures for the 3SAT problem, which come from the research literature of the past fifteen years [5,14,18,22,28, 29]. This incorporation of state-of-the-art research into the CS curriculum is desirable as it will better prepare CS graduates for careers in industry and graduate school.

Project number 5 asks the students to apply some of the algorithm design techniques that they learned in the 3SAT project sequence to another NP-complete problem. There are many possibilities for this project: The professor could present a problem and let the students come up with their algorithm, or they could find algorithms in the literature, or each student group can study and analyze one algorithm and present their work to the class, and so on. These address the third, fourth, fifth, sixth, and/or seventh targeted student outcomes.

We have anecdotal evidence that the student experiences with research, as described above, motivate some students to pursue research, and possibly careers as scientists--targeted student outcome number eight. Exposure to the proposed approach has helped to motivate several undergraduate students to pursue research on 3SAT. We have also been

able to develop an active SAT research group at our undergraduate institution. Several publications are likely to result from these efforts. Two students are pursuing graduate school admission, as well as minors or double majors to improve their chances for graduate funding. We believe similar success would be achieved by adopters of the proposed approach at other institutions.

#### **4 RESULTS OF USING THE PROJECTS IN THE AD&A COURSE**

We have used the project-based curriculum for NP-completeness in three offerings of our undergraduate AD&A course since 2003. As the approach matured, we have added more projects to make smaller incremental steps from one project to the next, or to introduce additional concepts that we think are important. The informal assessment conducted during these course offerings indicates that average undergraduate students are capable of completing the projects, and successful in understanding the underlying concepts. We found that most students were able to answer NP-completeness questions in quizzes and exams; in earlier offerings of the course these questions had appeared in homework assignments, or as extra credit exam questions that most students did not attempt. We covered more material after adopting the proposed curriculum than we had in previous offerings of the course, even if we disregard what the students learned about NP-completeness. It was easier to cover some topics: Branch- and- Bound, e.g., was trivial for the students after their implementation of a DLL solver. Every semester we dropped some minor topics that we had covered in the earlier versions of the course, but in all cases we believe that the students achieved all targeted student learning outcomes after implementing some variant of the proposed set of problems. We chose to exclusively assign programming projects based on SAT every time we have taught the AD&A course since 2003, partly because we wanted to experiment with this new approach; other adopters could certainly require programming projects from a combination of domains. We assigned non-programming take-home work on the topics that we had covered with programming assignments during earlier versions of the course, without significant loss of student comprehension. We found that the semester-long sequence of projects provides a nice thread for the course, that it was easier to motivate many of the topics that we have always covered, and that the students were more enthusiastic about the material.

Exposure to the proposed approach helped to motivate several undergraduate students to pursue research on 3SAT, and has allowed us to develop an active SAT research group at our undergraduate institution. The students have implemented a distributed 3SAT solver in Java and a fairly fast SAT solver in C++. They are creating a modular testbed to evaluate the performance of new algorithms and heuristics that they are designing. The students are contributing to the body of published

scientific work. A student poster presentation on a distributed 3SAT solver in Java was presented at a local symposium [citation removed], and a second student publication is under review. Several students plan to pursue graduate school directly after their bachelor's; they are pursuing publications, as well as minors or double majors, to improve their chances of obtaining a graduate assistantship or fellowship.

## 5 CONCLUSIONS AND FUTURE WORK

We have presented a novel project-based curriculum for teaching NP-completeness, classic algorithm design techniques, and techniques for solving intractable problems. The set of projects is based on the NP-complete 3SAT problem. The targeted student learning outcomes are to learn more about basic computability and NP-completeness, apply the scientific method, apply classic algorithm design techniques, implement algorithms for intractable problems using more advanced techniques, gain exposure to state of the art research, implement sophisticated algorithms and data structures, and apply algorithm design techniques that they have understood to a new domain. Many positive experiences from our use of this curriculum in our Algorithm Design and Analysis course indicate that further development and assessment of this approach are likely to be fruitful.

We plan to seek external funding to refine the student materials and develop faculty support materials to help others adopt the proposed curriculum, to formally assess the effectiveness of the approach and the materials, and to disseminate the products of these efforts locally, regionally, and nationally.

## 6 REFERENCES

- [1] ABET at <http://www.abet.org/>. Accessed 2005/11/19.
- [2] SAT Competition at <http://www.satcompetition.org/>. Accessed 2005/11/19.
- [3] A. Aho, J. Hopcroft, and J.D. Ullman. *The Design and Analysis of Computer Algorithms*. Addison-Wesley, 1974.
- [4] Sara Baase and Adam Van Gelder. *Computer Algorithms: Introduction to Design and Analysis*. Addison-Wesley, Third Edition, 1999.
- [5] A. Beringer, G. Aschemann, H. Hoos, M. Metzger, and A. Weiss. GSAT versus simulated annealing. In *Proceedings of the European Conference on Artificial Intelligence*, pages 130-134, 1994.

- [6] G. Brassard and P. Bratley. *Algorithmics: Theory and Practice*. Prentice Hall, 1996.
- [7] Jeremy Buhler and Martin Tompa. Finding motifs using random projections. In *RECOMB*, pages 69- 76, 2001.
- [8] Thomas Cormen, Charles Leiserson, and Ronald Rivest. *Introduction to Algorithms*. MIT Press, Second Edition, 2001.
- [9] James M. Crawford and Larry D. Auton. Experimental results on the crossover point in random 3-SAT. *Artificial Intelligence*, 81(1- 2):31- 57, 1996.
- [10] Martin Davis, George Logemann, and Donald Loveland. A machine program for theorem- proving. *Communications of the ACM*, 5(7):394- 397, 1962.
- [11] S. Donovan, J. Bransford, and J. Pellegrino. *How People Learn: Bridging Research and Practice*. National Academy Press, 2000.
- [12] Deborah East and Miroslaw Truszczynski. Propositional satisfiability in answer- set programming. *Lecture Notes in Computer Science*, 2174:138- 144, 2001.
- [13] Michael Garey and David Johnson. *Computers and Intractability: A Guide to the Theory of NP- Completeness*. W.H. Freeman and Co. 1979.
- [14] E. Goldberg and Y. Novikov. BerkMin: A fast and robust SAT solver. In *Design Automation and Test in Europe (DATE)*, pages 142- 149, 2002.
- [15] Oded Goldreich. *Modern Cryptography, Probabalistic Proofs and Pseudorandomness*. Springer Verlag, 1998.
- [16] Michael Goodrich and Roberto Tamassia. *Algorithm Design: Foundations, Analysis and Internet Examples*. John Wiley and Sons, 2001.
- [17] E. Horowitz, S. Sahni, and S. Rajasekaran. *Computer Algorithms*. Computer Science Press, New York, 1996.
- [18] Roberto J. Bayardo Jr. and Robert C. Schrag. Using CSP look- back techniques to solve real- world SAT instances. In *Proceedings of the Fourteenth National Conference on Aritificial Intelligence (AAAI'97)*, pages 203- 208, Providence, Rhode Island, 1997.

- [19] Anany Levitin. Do we teach the right algorithm design techniques? In *Proceedings of the thirteenth SIGCSE technical symposium on Computer Science education*, March 1999.
- [20] V. Marek, M. Dransfield, L. Liu and M. Truszczynski. Satisfiability and computing Van der Waerden numbers. *Lecture Notes in Computer Science*, 2919, 2003.
- [21] Udi Manber. *Introduction to Algorithms: A Creative Approach*. Addison- Wesley, 1989.
- [22] Matthew W. Moskewicz, Conor F. Madigan, Ying Zhao, Lintao Zhang, and Sharad Malik. Chaff: Engineering and Efficient SAT Solver. In *Proceedings fo the 38<sup>th</sup> Design Automation Conference (DAC'01)*, 2001.
- [23] Rajeev Motwani and Prabhakar Raghavan. *Randomized Algorithms*. Cambridge University Press, 1995.
- [24] Ketan Mulmuley. *Computational Geometry: An Introduction through Randomized Algorithms*. Prentice Hall, 1993.
- [25] Joint Task Force on Computing Curricula. *Computing Curricula 2001: Computer Science*. ACM and IEEE, 2001.
- [26] Steven Salzberg, David Searls, and Simon Kasif. *Computational Methods in Molecular Biology*. Elsevier Health Sciences, 1999.
- [27] Robert Sedgewick. *Algorithms*. Addison- Wesley, 1988.
- [28] Bart Selman, Henry A. Kautz, and Bram Cohen. Noise strategies for improving local search. In *Proceedings of the Twelfth National Conference on Artificial Intelligence (AAAI'94)*, pages 337- 343, Seattle, 1994.
- [29] Bart Selman, Hector J. Levesque, and D. Mitchell. A new method for solving hard satisfiability problems. In Paul Rosenbloom and Peter Szolovits, editors, *Proceedings of the Tenth National Conference on Artificial Intelligence*, pages 440- 446, Menlo Park, California, 1992. AAAI Press.
- [30] Clifford Schaffer. *A Practical Introduction to Data Structures and Algorithm Analysis*. Prentice Hall, Second Edition, 2001.
- [31] Mark Allen Weiss, *Data Structures and Algorithm Analysis in C++*. Addison Wesley, Second Edition, 1999.

- [32] Stephen A. Cook. The complexity of theorem proving procedures. In *Third Annual ACM Symposium on Theory of Computing*, pages 151-158, 1971.
- [33] Andrea F. Lobo, Ganesh R. Baliga, Seth Bergmann, Don Stone, and Anol Shah. Using Real-World Objects to Motivate OOP In a CS1 Lab. *Journal of Computing in Small Colleges*, 15(5):144- 156, 2000.
- [34] Don C. Stone, Seth Bergmann, Ganesh R. Baliga, A. Michael Berman, and John Schmalzel. A CS1 Maze Lab, using Joysticks and MIPPETs. In *Proceedings of the Thirtieth SIGCSE Technical Symposium of Computer Science Education*, pages 170- 173, 1999.
- [35] Gershon Zebovitz, Timothy Stackhouse, Matthew Williamson, Jonathan Feuss, and Andrea F. Lobo. A Distributed Davis-Putnam 3SAT Solver. Poster presentation at the *Rowan University STEM Symposium*, 2005.