Teaching Artificial Intelligence Across the Computer Science Curriculum using Sudoku as a Problem Domain

Jeffrey O. Pfaffmann and **William J. Collins**
Department of Computer Science
Lafayette College
Easton, PA 18042
pfaffmaj@cs.lafayette.edu and collinsw@lafayette.edu

Abstract

Artificial Intelligence (AI) is a field of both great breadth and depth. Thus, determining undergraduate material for an AI course can be problematic. Fortunately, AI is continually searching for new perspectives on problem solving that eventually propagate into the Computer Science mainstream. An approach is proposed for undergraduate AI education that utilizes these aspects of exploration and propagation. The approach introduces important individual techniques early in the computer science curriculum to form a foundation for the upper-level AI course focusing on research methods. This approach is explored using two Sudoku projects at different levels in the Computer Science curriculum, with constraint satisfaction used as the individual technique. Sudoku is a logic puzzle that has a great deal of appeal and is easily encoded as a constraint satisfaction problem domain. In the introductory-level CS 2 course, the Sudoku-based project uses a provided **BackTrack** class that can be employed to find a path through a maze, place eight queens, or schedule a knight's tour. This illustrates the power of recursion in general and backtracking in particular (a central aspect of constraint satisfaction techniques.) In the upper-level AI course, Sudoku is used as a problem domain for developing a puzzle solver using the full breadth of constraint satisfaction techniques and producing an optimal puzzle generator using a technique of the student's own selection. Students are required to write a research quality publication based on the results of their projects. The goal of the paper is to provide a research experience where not only a solution is derived but the reasoning process is directly explored. Due to the frequency that the AI course is taught, two separate student populations are discussed. The results from the dual population still allow for a viable exploration of this cross-curriculum approach. Finally, the paper illustrates that Sudoku is an excellent problem domain choice for teaching AI approaches in both introductory- and upper-level computer science courses.

Introduction

Artificial Intelligence (AI) has directly influenced many areas of computer science and produced a plethora of techniques. Thus, determining the important topics for a single semester undergraduate AI course is a non-trivial task. This issue is illustrated by the ACM Computing Curricula 2001 (CC2001) document, which details ten modules for Intelligent Systems study. Three of the ten modules are considered "core" and consist of 10 contact hours, with the topics being: Fundamental Issues, Search and Constraint Satisfaction, and Knowledge Representation and Reasoning. For the remaining "elective" contact hours, the material includes: Advance Search, Advanced Knowledge Representation, Agents, Natural Language Processing, Machine Learning and Neural Networks, Planning Systems, and Robotics. No single-semester course could cover this range of material in a rigorous fashion, leaving the question of what to cover. Generally the choice is left up to the faculty member's preference, which can be guided by a variety of factors.

We propose an approach that leverages the fact that AI is both an exploratory field that has also contributed directly to many areas of mainstream Computer Science. The proposal is to front-load core AI techniques in earlier low-level courses, where appropriate, with a brief discussion of how this technique is used in the higher-level AI course. This technique is reinforced with a project that ties the technique to the currently covered material. The "front-loading" of material serves to provide students with an intuition about AI techniques before they reach the actual course. Once reaching the AI course, the focus can switch from using techniques to developing techniques, or to teach students the process of exploration. To reinforce an exploratory approach students are required to develop a research quality paper that explores and extends a particular technique. To explore this approach constraint satisfaction techniques and the Sudoku problem domain were focused on. Constraint satisfaction was chosen because of its applicability to recursive solutions and thus is a good target for inclusion in a CS 2 course, in addition to being one of the core AI modules.

Sudoku is used for a variety of reasons: its symbolic

nature allows puzzles to be easily encoded as constraint satisfaction problems, puzzles exhibit a scalable complexity depending on the problem that is given, and these puzzles have a certain public popularity (Delahaye 2006). The literature has also suggested Sudoku as a good problem for course work (Edgington 2006) as well as providing several implementations, including: Python (Park 2006), Fortran (Metcalf 2006), and LaTeX (Wilson 2006). Constraint satisfaction researchers are also begining to use Sudoku as problem domain in their work (Simonis 2005).

Sudoku is a puzzle game where a player must fill in blank squares on a 9-by-9 grid so that certain requirements, or constraints, are maintained. The name is derived from a much longer Japanese phrase that translates to "*the digits must remain single*" (Wikipedia 2006). The goal of the game is to fill in every empty grid location with a number from 1 thru 9 so that the values in each column, row, and 3-by-3 subregion are unique. The subregions are determined by subdividing the 9-by-9 grid into 9 3-by-3 mini-grids, as shown in Figure 1. Sudoku puzzles are simple to understand but can be non-trivial to solve, making them a good problem domain for computer science education in general. This complexity is evident in the fact that there are approximately 6.671×10^{21} distinct Sudoku solutions (Felgenhauer & Jarvis 2006).

The Sudoku Puzzle

Sudoku, originally named Number Place, was created by Howard Garnes in 1979 and first published later that year in Dell Pencil and Puzzle Games. At first, interest in Sudoku was minimal until it reappeared in print and was given its current name by the Japanese publisher Nikoli in 1984. The internationalization of the craze began when Wayne Gould of Pappacom developed a program to create Sudoku puzzles and grade their level of difficulty (Shortz 2005).

	4		5		1	2		6
2				7	8	3		
					6		5	4
					5	1	2	8
	3						6	7
7	2	8	1					
8	7		9					
		6	2	5				9
9		3	6				1	

Figure 1: A sample Sudoku puzzle.

Figure 1 provides an example to examine how one would go about solving a Sudoku puzzle with pencil-and-paper. The puzzle starts with certain locations filled it, which can be a great number or very few depending on the puzzle complexity. Typically, a Sudoku puzzle will have a single solution, but if a puzzle is poorly designed then multiple solutions are possible, which is a feature that can be use as part of a class assignment.

There are a variety of logical methods for solving Sudoku puzzles. One such method is to examine a single grid location and determine all the possible values it may contain. If there is only one possible value then enter that value, otherwise note the possible values for future reference. For example, consider the possible values for the upper-left grid-location, or location (0, 0) if the row and column indexes start at 0. The only valid values for this location, given the current board configuration, is 3 by process of elimination. This can be determined by examining the location in column 0 and row 0, which are the values 1, 2, 4, 5, 6, 7, 8, and 9 but not 3. An alternate approach is to examine one particular value, for example 1, and determine where this value must be placed given the constraints imposed by the locations that already contain the same value. Using the example value 1, it can be seen that the upper-right subgrid does not contain a 1. Then by examining columns 6 and 7, it can be seen the subgrids below the upper-right subgrid contain a 1 in columns 6 and 7 allowing only column 8 to contain the value 1. Finally, column 8 in the upper-right subgrid can be seen to contain only one empty position, grid location (1, 8) that must then contain the value 1.

3	4	7	5	9	1	2	8	6
2	6	5	4	7	8	3	9	1
1	8	9	3	2	6	7	5	4
6	9	4	7	3	5	1	2	8
5	3	1	8	4	2	9	6	7
7	2	8	1	6	9	5	4	3
8	7	2	9	1	4	6	3	5
4	1	6	2	5	3	8	7	9
9	5	3	6	8	7	4	1	2

Figure 2: The solution to the Sudoku puzzle in Figure 1.

To solve most well-designed Sudoku puzzles, logic strategies can be used to propagate or resolve constraints imposed by currently existing values. But it is an open question whether all Sudoku puzzles with a unique solution can be solved by only constraint propagation, that is, without the use of backtracking. Mepham (Mepham 2005) has devised several "diabolical Sudoku" puzzles which, at least when they were released, could not be solved using known constraint propagation techniques and no backtracking. One such diabolical Sudoku puzzle is given in Figure 3.

For pencil-and-paper Sudoku solvers, backtracking is

1			9		7			3
	8						7	
		9				6		
		7	2		9	4		
4	1						9	5
		8	5		4	3		
		3				7		
	5						4	
2			8		6			9

Figure 3: A "diabolical Sudoku" puzzle, whose solution is unattainable using only (currently available) constraint propagation techniques.

unattractive as a single strategy, even for those with great patience and a large eraser. Some Sudoku puzzles that can be straightforwardly solved with backtracking require over 100,000 backtracks without constraint propagation or selection heuristics. But programmatically, the simplicity of backtracking makes it appealing, even without more complex constraint-satistfaction techniques. Students with only a minimal knowledge of Sudoku can develop a program to solve any Sudoku puzzle that has a unique solution. This ability illustrates the power of recursion in general and backtracking in particular.

CS 2 SUDOKU PROJECT

In the fall semester of 2005, the students in the CS 2 course at Lafayette College were assigned a programming project to develop a Sudoku-puzzle solver. In the course, there were 21 students: 15 in Electrical and Computer Engineering, 4 in Mathematics, and 2 in Computer Science. The language used in the class is Java. The project was assigned at the end of the chapter on recursion, and a case study of backtracking to find a path through a maze had been covered during class (Collins 2005). The following **BackTrack** class, based on a similar class in Noonan (Noonan 2000), utilizes a generalized backtracking algorithm from Wirth (Wirth 1976).

A user of the **BackTrack** class supplies:

1. a **Position** class to define what position means for this application (for example, in the maze case study, a position is simply a row and column);
2. a class that implements the **Application** interface; for the maze case study, the **Maze** class included the necessary details for finding a path;
3. a driver class to run the project, including input and output.

The same set up can be used for the Eight Queens problem and for the Knights Tour problem (each of which had been assigned in previous semesters). The fact that the **BackTrack** class and **Application** interface are re-usable helps students to appreciate backtracking as a general-purpose tool for solving a variety of problems.

```
import java.util.*;
public class BackTrack
{
   protected Application app;

   /**
   * Initializes this BackTrack object from an application.
   *
   * @param app the application
   */
   public BackTrack (Application app)
   {
     this.app = app;
   } // constructor

   /**
   * Attempts to reach a goal through a given position.
   *
   * @param pos the given position.
   *
   * @return true if the attempt succeeds;
   * otherwise, false.
   */
   public boolean tryToReachGoal (Position pos)
   {
     Iterator<Position> itr = app.iterator (pos);

     while (itr.hasNext())
     {
       pos = itr.next();
       if (app.isOK (pos))
       {
         app.markAsPossible (pos);
         if (app.isGoal (pos) || tryToReachGoal (pos))
           return true;
         app.markAsDeadEnd (pos);
       } // pos may be on a path to a goal
     } // while
     return false;
   } // method tryToReachGoal
} // class BackTrack
```

For the Sudoku project, students were required to use the **BackTrack** class and **Application** interface, as is. They had some flexibility with regard to the **Position** class: They could use the **Position** class from the maze project, or modify that class by adding a digit field (with values in the range from 0 through 9, inclusive) as well as row and column fields. Then an iteration over the positions accessible from a given position could simply iterate over the possible digits that the given position could take on.

The driver entailed a considerable amount of error-checking; for example, if the initial puzzle had two copies of the digit 7 in the same column, or if the puzzle

1	2	3						
4	5	6						
7	8		9					

Figure 4: An initial Sudoku configuration for which no solution is possible.

had no solution, as shown in Figure 4.

The project also had an extra-credit component: a bonus, worth 50% of the project grade, went to the first student whose project could detect if a puzzle had more than one solution. A sample program is available to instructors on request.

CS 2 RESULTS

Of the 21 students in the CS 2 course, 14 completed the project, and most of those utilized the hint of expanding the **Position** class to simplify the iterations. In anonymous comments, there was an even split among those who enjoyed the project, those who detested it, and those who were neutral. Also, two students complained that the **BackTrack** class and **Application** interface were too confining: They wanted the flexibility to develop an ad-hoc solution. Only four of the students in the class had heard of Sudoku before the project was assigned, and only two had played Sudoku.

AI SUDOKU PROJECT

In the fall semester of 2005 the Artificial Intelligence class (CS 420) was assigned a more expansive project, which consisted of three parts: a Sudoku solver, a Sudoku puzzle generator, and a 10-page conference quality paper based on the student's work. The class consisted of 16 students at either a junior or senior standing, with everyone having completed the core computer science curriculum. The Artificial Intelligence class is taught with weekly labs and two large projects that span 4-5 weeks. These large projects serve to provide students a research opportunity, thus some aspects were well specified while other aspects were more open-ended. One of the topics covered in Artificial Intelligence is constraint satisfaction problems, which is a *general-purpose* problem solving technique where the problem is formulated in terms of variables, the legal values for each variable, and the constraints between each variable. For Sudoku each grid location represents a variable, each variable will contain a value 1 through 9, and the value in each variable is regulated with three simple constraints.

For the Sudoku solver, the students were required to use constraint satisfaction techniques as discussed in class, which is also detailed in the class textbook by Russell and Norvig (Russell & Norvig 2003). The backtracking algorithm is common to the constraint satisfaction techniques presented in class, which is also similar to the **BackTrack** class presented above. The class text also introduces the idea of using the number of constraint validation checks performed as a metric of program efficiency, allowing the comparison of the different constraint satisfaction techniques. To implement this metric in the previously provided code, a counter would be added to the `isOK (pos)` method to track the number of times this method is called. Using this metric students could create a solver with the primary purpose of comparing different constraint satisfaction techniques. For the Sudoku project students were required to implement the following techniques: basic backtracking, backtracking with a variable or value ordering heuristic, forward check, forward checking with a variable or value ordering heuristic, and a constraint propagation technique. The resulting Sudoku puzzle solver program is a simple batch-program that accepts configuration parameters and a puzzle to solve. Once the program was complete, it produced a puzzle solution and printed the performance metrics.

Backtracking is included in the list of required techniques to provide a baseline for comparing the other techniques to. The remaining techniques are well known methods commonly associated with constraint satisfaction problem solvers. Variable and value ordering heuristics attempt to reduce the search space by selecting the variable or value that is most constrained and thus have the smallest branching factor. Forward checking maintains information about each unassigned variable and updates this information anytime a value is placed. If a locations become invalid then forward checking will detect the constraint violation and respond before basic backtracking would. Forward checking can be combined with variable and value ordering heuristics, with the *minimum remain values* heuristic (MRV) the most common. MRV is a variable selection heuristic where the variable with the fewest possible values is selected first, which is similar one of the paper-and-pencil techniques discussed previously. MRV is a good match with forward checking because the same information is computed for both, while producing a performance neither technique can produce alone. Constraint propagation is a technique where the constraints between the variables are searched to a limited depth to provide information for the basic backtracking to be performed more efficiently. Constraint propagation can be implemented as a preprocessor or integral part of the backtracking algorithm, with the later being a more complex design but a more effective approach. It should be noted that with the use of constraint propagation the processing time is shifted from the time searching the variable space to searching the constraint space, which will not show up in the required metric. Therefore, the

students were also encouraged to develop other metrics to determined their implementation's efficiency.

Once the solver is completed, the students perform experiments using several Sudoku puzzles of varying complexities to determine the general utility of the five different algorithms. Often in AI research, only part of the research effort involves producing a working solution, while the real work is in understanding why the system functions the way it does and how it relates to other solutions. Here students do this by comparing the different techniques to the basic backtracking approach and compare their implementations to the designs of others in the class.

For the second part, a Sudoku puzzle generator is required, which is the open-ended aspect of the project. Here students are required to take a Sukoku solution and produce a puzzle that has the most values removed and is still capable of producing only a single solution, or a minimum unique puzzle. The puzzle generator is a non-trivial problem that the students were given because there is no immediately obvious solution (other than brute force random value removal). The question then becomes, is there an elegant way of using the constraints to remove values from the puzzle solution so that the minimum sized puzzle is found that will still generate a single solution. It is also important to note that there will be multiple suboptimal results, and the problem becomes finding the global optimum (and whether or not there are several equivalent global optimums).

For the puzzle generator, the students were allowed to try any approach that they prefered, however, they were discouraged from using a purely brute force approach of randomly removing values. Students were also made aware that they would be rewarded for creativity and not punished for failure, that is, they were given room to explore without fear that they would loose points. This aspect of the Sudoku project seeks to stimulate the student's creativity, or the ability to try new avenues, which is an important aspect of any research experience.

The final part of the project, students are required to produce a 10-page conference quality research paper, using the ACM special interest group formatting. In the paper, students discussed their methodology, algorithm implementation, results and lessons learned. Students at Lafayette College do a great deal of writing, but much of it is outside of the department and does not follow the format that is common for their chosen field of study. One goal of the Artificial Intelligence class is to give students the opportunity to become aquatinted with a style that is common to computer science. To facilitate this goal, students were required to use templates provided by the ACM for conference proceedings. Additionally the students were required to minimally have the following sections.

- **Abstract** – The paper should start with a 500-word abstract summarizing the work.
- **Introduction** – Introduce the problem and approach to a casual reader with a computer science background.
- **Formal Description** – Formally describe the problem and the algorithms used to solve it.
- **System Performance** – Describe project methodology for determining system performance and quantitatively discuss its performance.
- **Conclusion** – Reintroduce the problem/approach, qualitatively discuss the performance of the system, and draw-out any conclusions or lessons that may have been learned.

Students were also provided example constraint satisfaction research papers, as well as, being encouraged search out their own examples. Students were graded primarily on publication quality, as is any researcher who is submitting an article to a conference. The condition of the source code, i.e., formating and comments, were expected to adhere to a certain standard, which could negatively impact a student's grade if these standards were not maintained. Furthermore, non-functioning and incorrectly-functioning programs could also negatively impact a student's project grade. Again, the goal of the project is to provide students with a research experience where the student develops a software infrastructure, explores new problem solving directions, and generates a document to summarize the results. In future versions of this class a double-blind review process will be added, where each student will be given two submissions that they must review, comment, and accept or reject for a hypothetical conference.

AI Project RESULTS

Of the 16 students, all but two students submitted a complete article with correct fully functioning programs. For the two students, one submitted a correct sudoku puzzle solver and generator, while the other submitted only a puzzle solver. Each puzzle solver was run with a common set of five different Sudoku puzzles and the metrics were compared. These metrics were presented to the class and discussed during a class period reserved for recapping the project experience. As would be expected, the metrics derived from each program varied greatly for the techniques that extended the basic backtracking algorithm, while the metrics for backtracking alone typically were the same.

The puzzle generator results varied greatly, with none of the submitted solutions able to produce the minimum unique puzzle. The bulk of the students used an optimized value removal technique, where they would select variables based on some heuristic then run their solver to see if the solution was still unique. The remainder of the students used stochastic techniques, such as genetic algorithms or simulated annealing. This second body of students consistently obtained better results than did the students using removal techniques. But again, the second group also used their solver to detect if the puzzle generated the same unique solution. It should be

noted that this technique is error prone because multiple solutions might not be detected, but given the class time-line this was an acceptable approach.

The greatest project success was the general high quality of the submitted articles, which typically came closer to a conference quality paper than what might be expected for a first attempt at formulating such a document. Overall, students were able to assimilate the lessons learned from their previous courses in algorithms and statistics, critically examine their own work, and presenting that work in the correct light. From a formatting perspective the articles were typically well organized and presented in a way the reader could easily understand what the author had done. At this point the impact of introducing backtracking in CS 2 could not be evaluated in terms of how that influence the AI project, as these courses were held in parallel. But this information will make itself available during the Fall 2007 semester when the Artificial Intelligence course is offered again.

CONCLUSIONS

In the CS 2 course, the goal of the project was to illustrate the generality of backtracking as a problem-solving tool. Students with a minimal knowledge of Sudoku were able to solve any Sudoku puzzle by utilizing an already available **BackTrack** class, an appropriate implementation of the **Application** interface, a fairly small **Position** class, and a driver. As such, the project was a success. For the two students who found the required framework constraining, there was not much recourse: An ad-hoc solution without backtracking would not have met the goal of the project, would not have allowed the solution of "diabolical" Sudoku puzzles, and would have either required a substantial knowledge of Sudoku or tempted students to scour the Web for an already existing program.

For the Artificial Intelligence course the goal is to use Sudoku as a problem domain for a constraint satisfaction project where the students can have a real research experience. They were required to create a puzzle solver using a variety of constraint satisfaction techniques and a puzzle generator using a technique of their own determination. Then using the solver and generator results, students wrote a conference quality research paper. In the paper, the solver provided a system where students needed to explain why it worked the way it did, and the generator provided an open-ended task where students could explore the creative process inherent in research. Based on paper quality, the project achieved its goals by forcing the students to begin thinking in a research mode, as opposed to an assignment mode. This was especially seen in the lab, where students began to have last minute "revelations" that would cause students to begin new experiments on an algorithm variation for additional results.

These results show Sudoku is a very natural fit for backtracking and associated constraint satisfaction techniques. Additionally, the presented results indicate Sudoku is well suited for teaching recursion in the introductory computer science curriculum, as well as, providing upper-level students with a solid research experience in Artificial Intelligence courses. Thus, the two presented Sudoku problems illustrate how Artificial Intelligence techniques can be taught across the Computer Science curriculum, allowing for students to understand the broad issues of an expansive field. Finally, it is expected that these results will be further supported in the next iteration of the uppper-level AI course. For this later course, the students will consist of those exposed to constraint satisfaction in the lower-level CS 2 course.

References

CC2001. Computing Curricula 2001: Computer science report of the joint task force on computing curricula. http://www.acm.org/education/curricula.html.

Collins, W. J. 2005. *Data Structures and the Java Collections Framework*. New York, NY: McGraw-Hill, 2nd edition.

Delahaye, J.-P. 2006. The science behind Sudoku. *Scientific American* 80–87.

Edgington, J. 2006. Solving Sudoku puzzles: Nifty course assignments. *J. Comput. Small Coll.* 21(4):90–91.

Felgenhauer, B., and Jarvis, F. 2006. Mathematics of Sudoku I. *Mathematical Spectrum* 39(1):15–22.

Mepham, M. 2005. Farewell to ariadne′s thread. http://www.sudoku.org.uk/goodbye.htm.

Metcalf, M. 2006. A Sudoku program in Fortran 95. *SIGPLAN Fortran Forum* 25(1):4–7.

Noonan, R. 2000. An Object-Oriented view of backtracking. In *Proceedings of the 31st SIGCSE Technical Symposium on Computer Science Education*, 362–366.

Park, C. 2006. Why not Python?, part 2. http://www.linuxjournal.com/article/8729.

Russell, S., and Norvig, P. 2003. *Aritificial Intelligence: A Modern Approach*. Englewood Cliffs, NJ: Prentice-Hall, Inc., 2/e edition.

Shortz, W. 2005. *Sudoku Easy Presented by Will Shortz*. New York, NY: St. Martin's Griffin.

Simonis, H. 2005. Sudoku as a constraint problem. In Hnich, B.; Prosser, P.; and Smith, B., eds., *Proc. 4th Int. Works. Modelling and Reformulating Constraint Satisfaction Problems*, 13–27.

Wikipedia. 2006. Sudoku entry. http://en.wikipedia.org/wiki/Sudoku.

Wilson, P. 2006. Sudoku bundle for displaying, solving and generating Sudoku puzzles. http://www.ctan.org/tex-archive/macros/latex/contrib/sudokubundle/sudokubundle.pdf.

Wirth, N. 1976. *Algorithms + Data Structures = Programs*. Englewood Cliffs, NJ: Prentice-Hall, Inc.