COMAP 2008
Mathematical Contest in Modeling

Team 3825

Problem B: Creating Sudoku Puzzles

February 18, 2008

Contents

1 Introduction

To make a Sudoku-generating algorithm that also produces a puzzle of specified difficulty, we first determined how our metric would be accurately rate difficulty. It was immediately obvious that the sparseness of a Sudoku puzzle does not necessarily determine its logical difficulty. We then researched the various methods used to solve Sudokus and were able to identify how advanced each method was based on their logical complexity in comparison with one another.

After ranking the complexity of the methods, we researched and found a code by the name of 'Dancing Links' that was capable of generating a Sudoku and determining its uniqueness. By combining our methods with the Dancing Links algorithm, we produced a generator that could generate a Sudoku that was guaranteed to be both unique and of a given difficulty level. This was achieved by having our "hybrid" algorithm generate a puzzle, and remove cells one by one, each time check if the puzzled maintained uniqueness, or if the difficulty had yet exceeded that which was selected.

In this way, we found a metric by which we could generate and rate Sudoku puzzles, that was not only simple, but gives ratings representative of the level of logic need to solve them.

2 Definitions

This terminology will be used throughout the paper.

2.1 Basic Definitions

- A **board** (also: Sudoku, puzzle) is a 9 x 9 grid containing initially filled cells which imply a unique solution
- A **solution** (also: Completed Puzzle) is a completely filled board
- A **non-unique puzzle** is a puzzle with more than one solution. They will inherently require trial-and-error (with lack of error) methods to solve.
- A **cell** is one of 81 small grid elements
- A **filled cell** is a cell with a presumably correct value in it
- A **value** (also: an entry) is the number in a filled cell
- A **candidate** is a possible digit that may be placed into a cell. Usually denoted by 'small numbers.'
- A **grouping** is a collection of 9 cells, which upon puzzle completion, contain 1-9 inclusive. There are three kinds of groupings on a standard Sudoku board: by row, by column and by box.

- A **row** is one of nine 1 x 9 rectangular horizontal groupings
- A **column** is one of nine 9 x 1 rectangular vertical groupings
- A **box** (also: square) is one of nine 3 x 3 square groupings
- A **method** is a logic based strategy used to solve a puzzle
- A **cancelation** is the removal of a candidate from a cell.
- **Elimination** is the process of removing cells from a solution to create a puzzle. All examples of elimination in this project avoid removing cells that would cause a puzzle non-unique.
- A **Max Eliminated** puzzled is one such that upon removal of any filled cells, the resulting puzzle is non-unique.

2.2 Methods

Sudoku solvers employ a wide variety of methods when solving puzzles. A number are listed below; most with examples. We will be discussing these methods throughout the project as they are fundamental to our generator algorithm and difficulty metric.

2.2.1 Naked Singles

Any given cell where there exists only one candidate is filled in with that candidate.

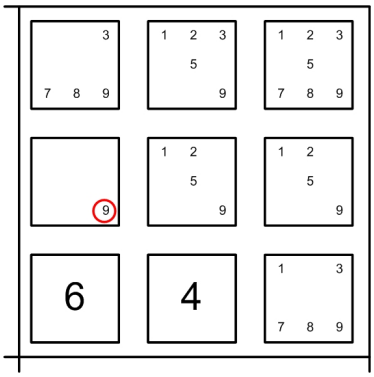

Figure 1: An example of a Naked Single. Here the circled 9 is the only candidate for the given cell.

2.2.2 Hidden Singles

If any cell within a given grouping is the only possible location for a given candidate, then that cell is filled in with that candidate.

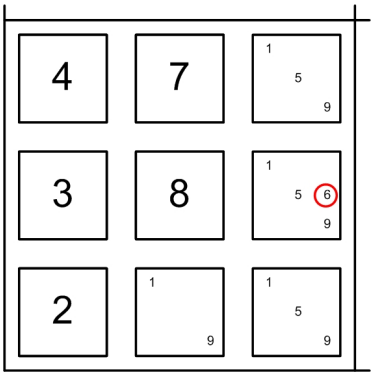

Figure 2: An example of a Hidden Single. Here the circled 6 is the only candidate 6 within the given grouping.

2.2.3 Locked Candidates

There are two ways Locked Candidates can occur.

- When a candidate for a given box appears only in a single row/column, the candidate(s) outside the box and inside the row/column can be eliminated.

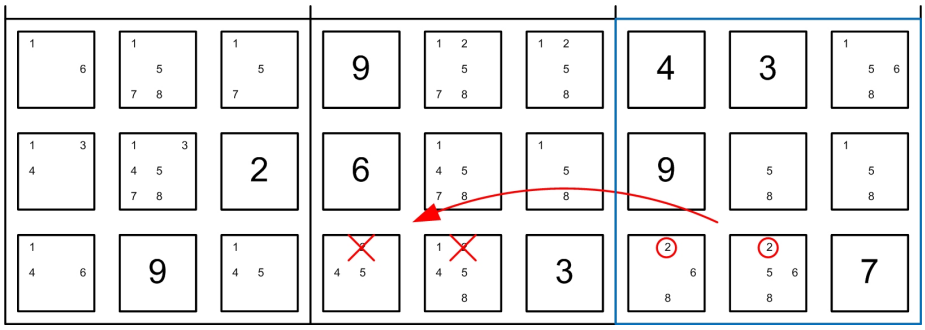

Figure 3: An example of a type 1 Locked Candidate. Here the candidate 2's of the rightmost box appear only in the bottom row (circled). Therefore, no candidate 2's can exist outside the rightmost box when in the bottom row.

- When a candidate for a given row/column appears only in a single box, the candidate(s) outside the row/column and inside the box can be eliminated.

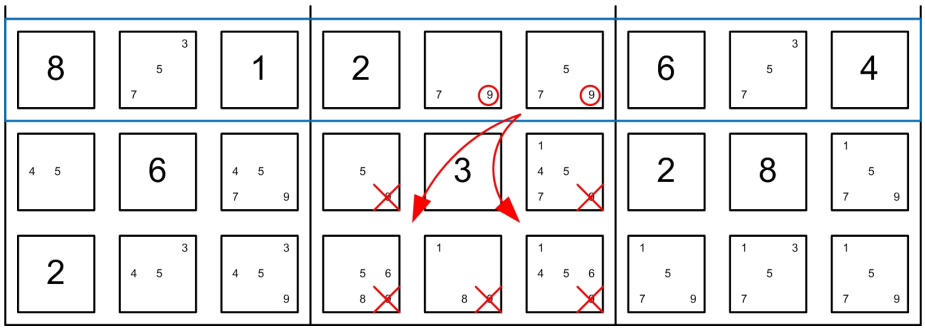

Figure 4: An example of a type 2 Locked Candidate. Here the candidate 9's of the top row appear only in the middle box (circled). Therefore, no candidate 9's can exist outside the top row inside the middle box.

2.2.4 Naked Pairs

When two unfilled cells within a single grouping contain the same two (and only two) candidates, no other cells within that grouping can contain those candidates.

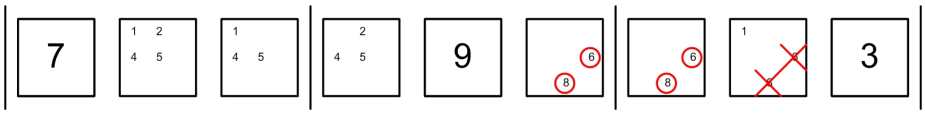

Figure 5: An example of a Naked Pair. Here, the circled 6's and 8's appear naked in this grouping. Therefore, candidate 6's and 8's cannot exist elsewhere in this grouping.

2.2.5 Hidden Pairs

When two unfilled cells within the same grouping contain contain the only two instances of two given candidates within that grouping, all other candidates in the two cells can be eliminated.

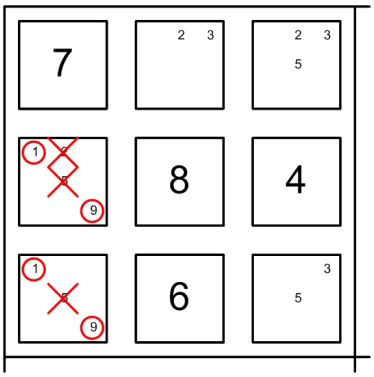

Figure 6: An example of a Hidden Pair. Here the candidate 1's and 9's appear in only two cells of this grouping (circled). Therefore, no other candidates can exist in these two cells.

2.2.6 Naked/Hidden Triples, Naked/Hidden Quads

These are straightforward generalizations of the single and pair strategies listed above.

2.2.7 Expert Methods

Several more advanced strategies exist which look at complex relations between elements across many different groupings. Some of these strategies include Fish methods. Though there are more advanced methods still, we feel these methods are representative of the most used. Fish methods work on this principle:

> *Identify N columns (2 for X-wing, 3 for the Swordfish, 4 for a Jellyfish, 5 for a Squirmbag) with two to N candidate cells for a specific value (the defining cells). If the defining cells fall on exactly N rows, then in all N rows, the candidates are canceled from any cell that isn't one of the defining cells. These methods can also be done with rows and columns interchanged.* [1]

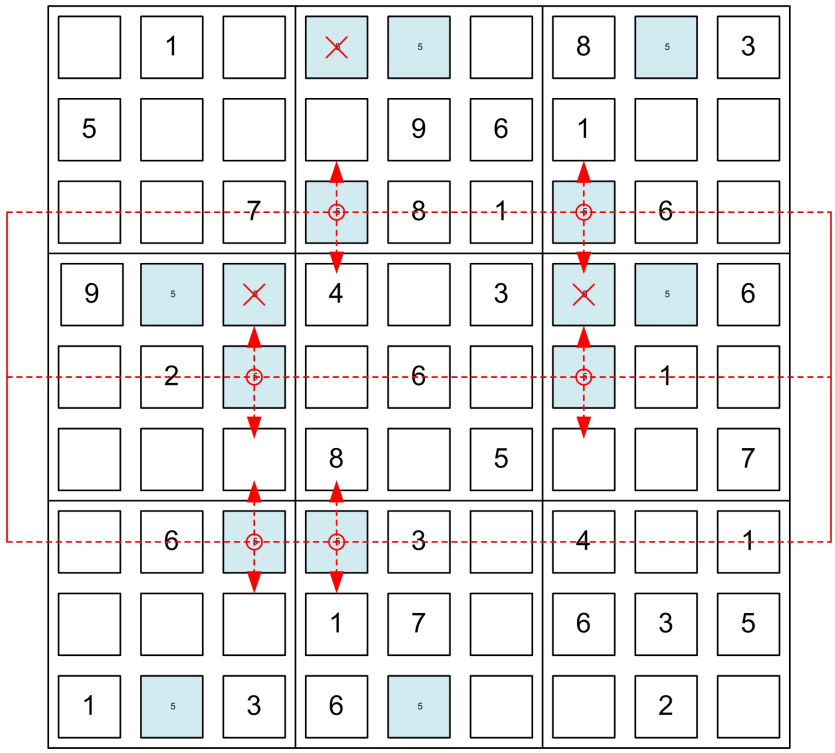

Figure 7: A demonstration of the Swordfish method (N=3). All candidates other than 5's have been hidden, and candidate 5 cells have been shaded blue. Here, N=3 rows all contain two to N candidate 5's (for this case, all contain two). Because these defining cells fall into N=3 columns, all 5's in these columns not in a defining cell are canceled.

2.2.8 Trial and Error Methods

There are a lot of names for these methods, such as Nishio, Bifurcation and Ariadne's Thread, but they all have the same principle [7]. If other methods have been exhausted, then there may be no other choice than to assign a cell a value and continue on this assumption until a contradiction is reached or the puzzle is solved. All possibilities are explored until the solution is found.

3 Difficulty Metrics

3.1 Sparseness

This rating metric, common among computer generated puzzles, uses the initial number of vacant to rank its difficulty [3, 4]. Although simple, it does not rec-

ognize that difficulty and sparseness of initially filled cells are often independent of one another. While sparser puzzles may require more steps to complete, they are not necessarily more difficult. This is demonstrated by Fig. 8(a), which while being thinly populated, can be solved using only Naked and Hidden Singles. It is also possible to have more populated puzzles which are impossible to solve without more difficult methods such Locked Candidates and Hidden Pairs, such as in Fig. 8(b). Puzzles generated in this manner cannot be tailored to higher level methods.

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

(a) A sparse but easy puzzle.

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

(b) A populated but relatively difficult puzzle.

Figure 8: Examples showing incorrectness of using sparseness as a difficulty metric.

3.2 Elapsed time of Expert

Difficulty could potentially be measured by a completion times averaged over a number of experts. Ideally, this number would be large enough to eliminate subjectivity. Computers can only roughly approximate the thought process of an expert and this process is by no means simple, as experts oftentimes deviate from recursive strategies, and it difficult to equate computer solving time with expert solving time. Thus, expert time is an impractical approach for generator that employs a user specified difficulty.

3.3 Weighted sum of methods used

Another metric attempts to quantify the effort employed when solving a puzzle. A puzzle which uses a more difficult method multiple times will inherently take more effort than a puzzle that uses it only once. A possible system of ranking difficulty addresses this by assigning a number of points to each method. Whenever a method is used, its points are added to a counter. When the puzzle is solved, the total is assigned a difficulty based on the range it falls in. This idea has been advanced to assigning the first use of a method more points than

subsequent uses to add a 'humanness' factor [6]. Although sensible, this algorithm is complicated. Like the previous method, it also has serious issues with subjectivity. Without concrete research into the time or effort expended by the average Sudoku player when executing a method, points cannot be assigned without some degree of arbitration.

3.4 Level of Required Skill

The fourth possible way of determining difficulty is simply what methods are necessary to solve the puzzle. The higher the difficulty of the methods required, the higher the difficulty of the puzzle. This metric maintains objectivity more so than the previous by saying 'this method is more difficult' rather than 'this method is *x times more* difficult. Additionally it is much simpler than the other two methods that let level required solving logic have some say in how difficult it is.

This is the metric we chose. We grouped varying methods based on how subtle or difficult to learn the methods are, and rated puzzles based on the highest level methods necessary to solve them. The application of this metric will be discussed in the next section.

4 Our Approach

To implement our difficulty metric, it is necessary to rank the strategies we have listed in Section 2 by their complexity. Once this is done, a puzzle is assigned a difficulty based on the strategies required to solve it. This, with some trial and error when generating solutions, gives us all we need to generate a puzzle of any requested difficulty.

4.1 Difficulty Hierarchy

The methods are divided into a hierarchy of 5 levels based on how difficult they are to use and/or learn.

4.1.1 Difficulty Level 1

The most basic methods are Naked Singles and Hidden Singles. These simple methods use the inherent exclusion principles to determine if the number contained within the cell in question can only have one value, or if a particular cell is the only location in which a certain value can be placed.

4.1.2 Difficulty Level 2

The 2^{nd} Level of puzzles require Locked Candidates to be solved. This method moves beyond finding cancelations resulting from observations of explicit infor-

mation (deductive logic) to finding cancelations that are not immediately visible (inductive logic).

4.1.3 Difficulty Level 3

Level 3 methods are the Naked and Hidden Pairs, Triples and Quads. Because these methods (more commonly than second level methods) do not result in a cell being solved, they are grouped as Level 3. Triples and Quads, while not as simple as Pairs, are reasonable logic extensions thereof and are thus included in the 3^{rd} level as well.

4.1.4 Difficulty Level 4

Even more complicated are the Fish and other Expert methods, which often use the entire board to eliminate candidates. Further still, they do not usually result in a filled cell.

4.1.5 Difficulty Level 5

The 5^{th} level of Sudoku are those that cannot be solved using any of the methods described above. They must be completed using trial and error-methods, and due to the nature of the algorithm, are always Max Eliminated. A limitation of our algorithm is that does not know what solutions which, when Max Eliminated, require trial-and-error methods. As a result the algorithm could potentially start over with a number of times before it reaches a puzzle that is 5^{th} level.

4.2 Preliminaries for Algorithm Implementation

Our algorithm requires a solver that will be able to tell us whether or not a given Sudoku board can be solved uniquely. To do this we could, for example, use the Dancing Links Algorithm as it was coded by Per-Anders Ekström [3]. Dancing Links is an application of Donald Knuth's Algorithm X, which is capable of solving any exact cover problem [5]. An exact cover problem asks for a partition of a set, if one exists. As it is an exact cover problem, Sudoku can be solved using the Dancing Links algorithm. We refer to the Dancing Links solver as `dlxsolve()` in the description below.

4.3 Sudoku Generation Algorithm

The user specifies the difficulty level with the command >>`sudoku(level)`. This chooses a particular recursive solving algorithm.

The solution is generated first. This ensures every puzzle is consistent. The first row is randomly assigned values 1 thru 9 inclusive. The rest of the solution is generated randomly through use of `dlxsolve()`.

Puzzles are then generated by randomly choosing a cell and eliminating it if in-so-doing the puzzle remains unique. In parallel, indices of successful and attempted eliminations are removed from vector `stillin`. Elimination continues until the chosen algorithm is incapable of solving. If `stillin` is emptied before the chosen algorithm is unable to solve it, a new solution is generated in its place and `stillin` is reset. The second to last iteration (still solvable by the specified algorithm) is supplied as the puzzle and is thusly of maximum difficulty whilst remaining within the desired level. Level 5 puzzles are simply Max Eliminated puzzles that cannot be solved by the Level 4 algorithm.

4.4 Error Analysis

Although an error analysis could not be conducted within the timeframe of this project, it could be done with relative ease. In order to test our metric for errors, we would run a large sample of published puzzles from newspapers and books through our algorithm and determine their with respect to our metric.

Because the rating standards employed by publishers are more stringent than those by used the majority of online generators (which most often use sparseness as a metric), their ratings are much more accurate. We could statistically analyze how closely our ratings compare against the ratings produced by professional sources.

5 Conclusion

The presented problem was to create an algorithm capable of generating a unique sudoku puzzle and to determining the puzzle's difficulty from a range of at least four classifications, while keeping the algorithm as simple as possible. Using the Dancing Links algorithm coded by Per-Anders Ekström as a Sudoku generator, we developed code that would be able to produce a Sudoku of any one of five difficulty levels, which are determined by the methods necessary to solve the puzzle.

The algorithm we conceived uses well-known methods of solving Sudokus, ordered according to logical complexity, to determine how hard a given Sudoku is. By using this metric in combination with the Dancing Links algorithm, we are able to recursively determine if a possible puzzle is of the desired difficulty before the generation process is finished. The process can then either continue the generation if the puzzle is within the proper difficulty or backtrack if the puzzle exceed the bounds.

The various other metrics for determining difficulty were discarded, either because they were too simple, too complex, or overly arbitrary and subjective. The Sparseness metric, while it gives the impression of difficulty through the

amount of time and effort spent searching for clues, cannot guarantee the generation of a truly more difficult Sudoku. The Expert Time metric is overly complex and too difficult to measure accurately. The Sum of Weighted Methods metric is the most likely candidate, behind the one we actually chose, but the points assigned to the various methods are arbitrary and subjective, leaving the values of these methods up to the personal preferences of the rater.

Our chosen metric is able to define a puzzle's difficulty based on how advanced a Sudoku player's methods must be in order to solve it. While the ranking of the methods may seem somewhat arbitrary, the reasoning behind our assignations is very logical.

Determining a Sudoku's difficulty by what methods are needed to solve it proves to be the most logical choice when attempting to rate a Sudoku with a clear, simple, and accurate metric. Additionally, our metric can easily be extended to include methods and levels of difficulty not described in this project. Our metric, along with Dancing Links and our own algorithm described in Section 3.3, provides a simple method for generating a Sudoku puzzle of any desired difficulty.

References

[1] Sudoku Solver by Andrew Stuart. Stuart, Andrew.
http://scanraid.com/sudoku.htm. 2008.

[2] Solving Sudoku. Johnson, Angus.
http://www.angusj.com/sudoku/hints.php. 2005.

[3] Sudoku (Dancing Links Solver). Ekström, Per-Anders.
http://www.mathworks.com/matlabcentral/fileexchange/loadFile.do?objectId=14073&objectType=File. 2/23/2007.
Code acquired from MATLAB Central File Exchange.

[4] Sudoku_lite. Fasino, Dario. http://www.mathworks.com/matlabcentral/fileexchange/loadFile.do?objectId=9462&objectType=File.
12/27/2005.
Code acquired from MATLAB Central File Exchange.

[5] Dancing Links. Knuth, Donald. *Millenial Perspectives in Computer Science*. 187-214. 2000.

[6] Sudoku Programmers Forum Discussion. Various.
http://www.setbb.com/phpbb/viewtopic.php?t=142&mforum=sudoku
2005.

[7] Sudopedia. Various.
http://www.sudopedia.org/wiki/Main_Page. 2007.