Solving Futoshiki Puzzles with Backtracking CSP Search

Team Members:

Steve Lu : g3lustev

Wenfeng Jiang : g5jiangw

Type: CSP

Introduction

When it comes to logic puzzles, Sudoku definitely stands at the forefront of popularity. However some may know of a similar but different game, Futoshiki. Despite being lesser in popularity, it has earned itself a consistent player base and a weekly spot in some newspapers.[1] It does not seem like there are many Futoshiki solving programs, and so we have created an AI that solves Futoshiki games using backtracking CSP search.

Futoshiki is similar to Sudoku in the way that you must fill a square grid with numbers such that the rows and columns do not contain repeats. Also the range of numbers that can fill the grid is the same as Sudoku—a five by five grid means each row and column must have all the numbers from one to five. On the other hand, unlike Sudoku, there are randomly placed greater-than and less-than signs between cells. The player must also, along with ensuring that every row and column is clear of duplicates, fulfill these inequality constraints.

The Problem

The problem we are trying to solve is the game itself. Using constraint satisfaction we aim to build a program that can find the solution for any game of Futoshiki.

We chose constraint satisfaction for solving Futoshiki problems because a logic puzzle like Futoshiki is exactly that, a set of constraints that the player must satisfy. To be specific, there is a NotEqual constraint over all pairs of cells in every row and column and an X-GreaterThan-Y or X-LessThan-Y constraint on all the pairs of cells with a greater-than or less-than sign between them. Each Futoshiki game has a unique solution and the purpose of the game is to solve it without guessing or probability, so that removes using heuristic search or Bayes nets from the equation. CSP search is essentially eliminating all incorrect answers until only perfect ones remain, so it is without a doubt the most appropriate technique.

Program Evaluation

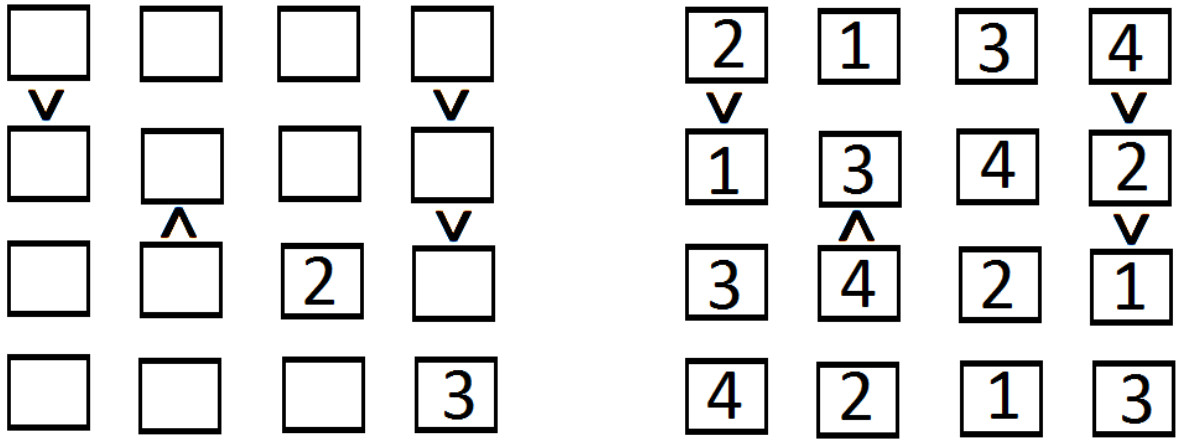
Our program takes the given board and gets the inequality constraints and any already assigned cell values. It enforces a NotEqual constraint on every pair of cells in the same row and every

pair in the same column. Furthermore, any inequality constraints are enforced on their corresponding pairs of cells. A propagator constantly checks ahead of time and removes any domain values that won't satisfy the constraints. If the current solution path cannot satisfy all the constraints the program backtracks and tries another solution path.

To test its validity we ran it on five different, yet increasingly difficult Futoshiki puzzles from "https://www.brainbashers.com/futoshiki.asp.".

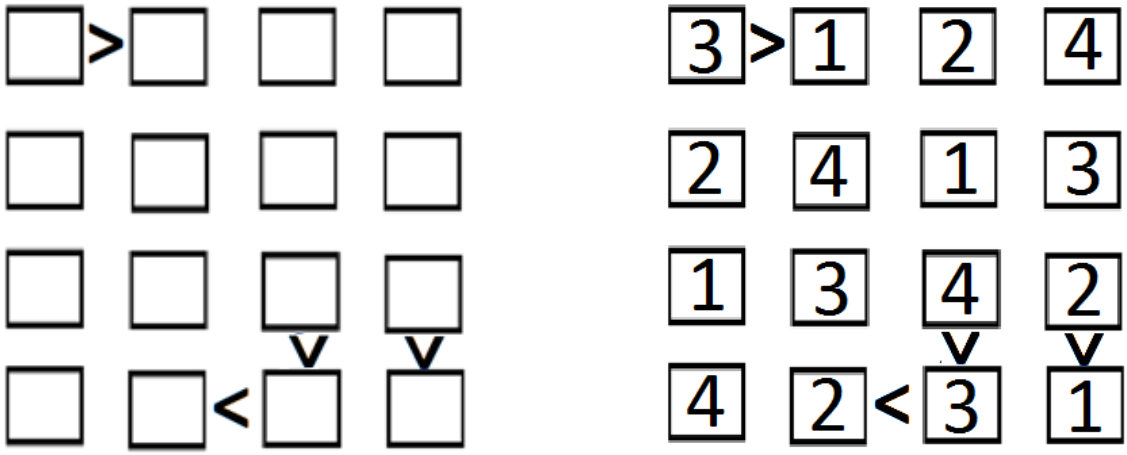
We started off with an "easy" 4 x 4 puzzle. It correctly solved it with a CPU time of 0.015600100000000006s.

The search made 16 variable assignments and pruned 42 variable values.



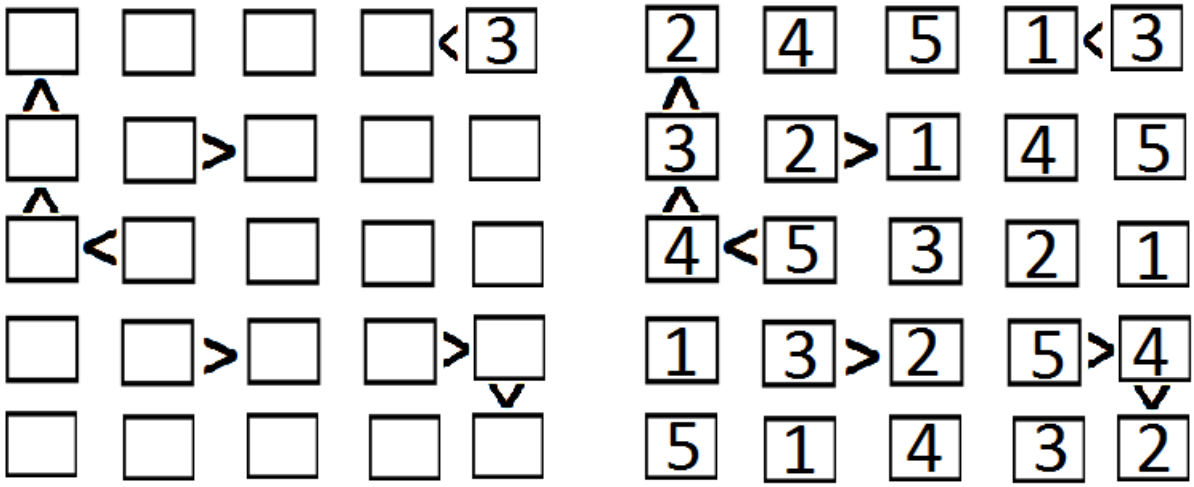
Next we did a "hard" 4 x 4 puzzle. It correctly solved it with a CPU time of 0.015600100000000006s, which is the same as the easy puzzle.

The search made 17 variable assignments and pruned 81 variable values.



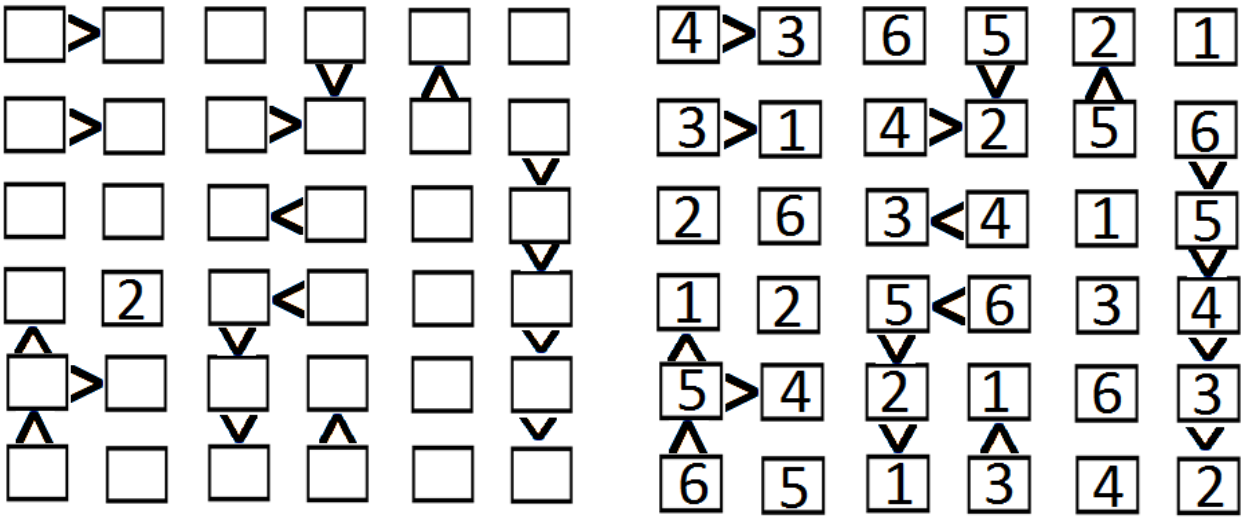
We started increasing the grid size with this “hard” 5 x 5 puzzle. It still solved correctly with a CPU time of 0.07800050000000003s.

The search made 33 variable assignments and pruned 207 variable values.



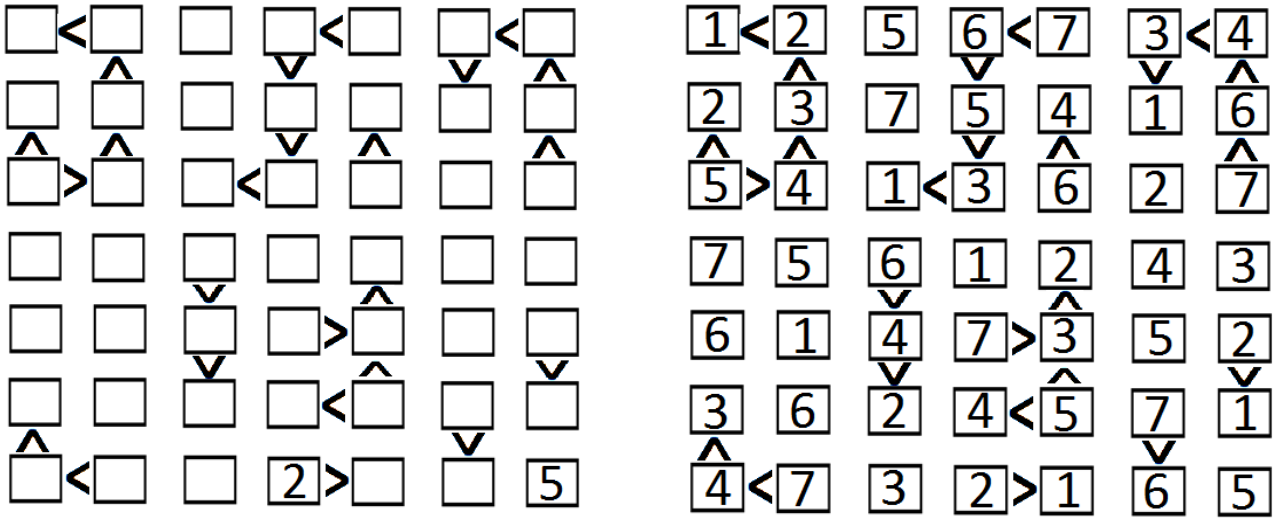
Then we tried a “hard” 6 x 6 puzzle. It correctly solved with a CPU time of 0.6864043999999999s.

The search made 153 variable assignments and pruned 1464 variable values.



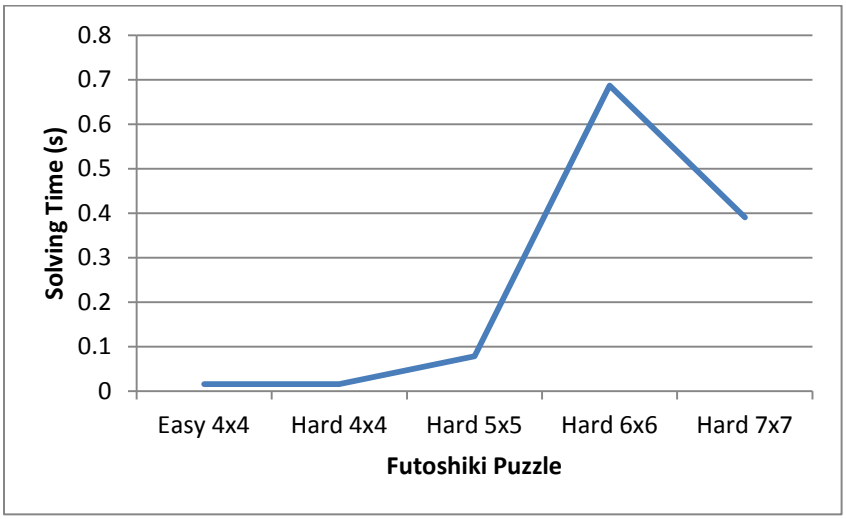
Finally, we tested our program on a “hard” 7 x 7 puzzle. It correctly solved it with a CPU time of 0.39000250000000003s.

The search made 69 variable assignments and pruned 503 variable values.



Analysis

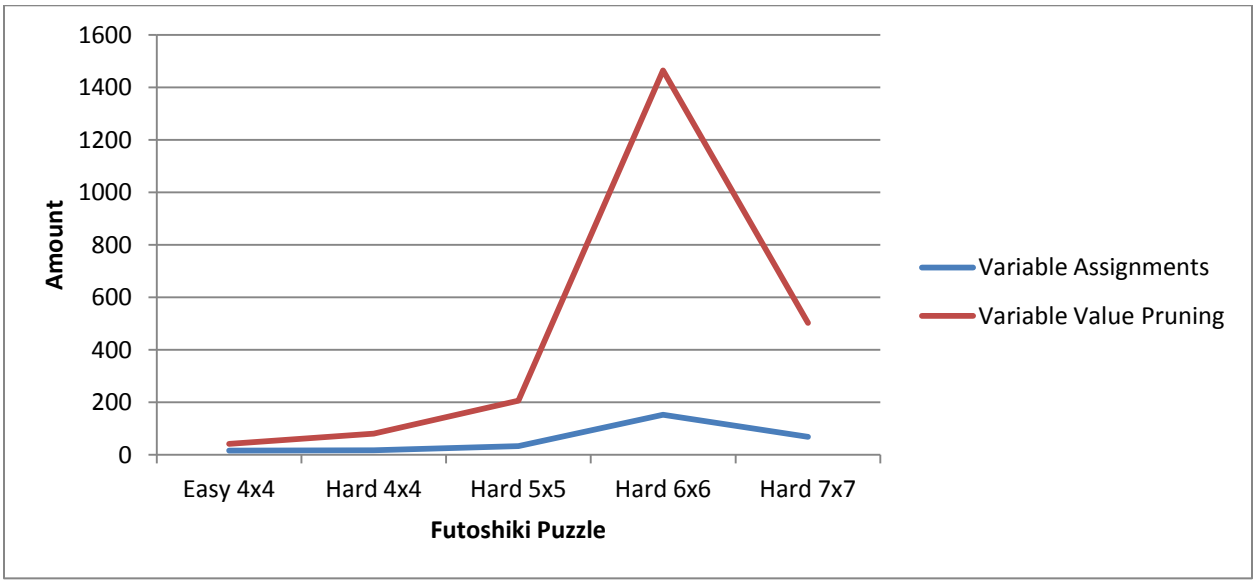
This is a graph comparing the time it took for the program to solve each puzzle.



In general, it seems that the larger the grid for the puzzle, the longer it takes the program to solve. However, the outlier is that the 6 x 6 grid took longer to solve than the 7 x 7, so grid size may not be everything.

The time increase for increasing the grid length and width by one is exponential, so it is likely that for enormous Futoshiki puzzles the solving time could be substantial. On the other hand, it is also probable that Futoshiki puzzles large enough to pose a problem to the program are rarely, if ever created. Factoring in that every puzzle we tested on was solved in under a second, we can conclude that our program is a success in terms of time efficiency.

Now let us compare variable assignments and variable value pruning.



Both variable assignments and value pruning followed the same trend as solving time, which makes it very likely that the 6 x 6 puzzle, despite being smaller than the 7 x 7 puzzle, was more difficult. If we assume this to be true, then it is apparent that our program works harder for more difficult puzzles. However, we do not know if difficulty for our program is the same as difficulty for humans. It is generally thought that a larger puzzle is more difficult for people to solve, and it is good if it is true that humans would struggle with the 7 x 7 puzzle more than the 6 x 6. This is because it means our program has the potential to solve puzzles that humans find overwhelmingly difficult since it does not follow human-perceived difficulty.

Another interesting thing to note is that the amount of variable values pruned for the 5 x 5 puzzle is not much lower than the amount for the 7 x 7 puzzle. Thus, it is possible that the amount for a 10 x 10 puzzle is not much higher than the 7 x 7 or even lower since the 7 x 7 is lower than the 6 x 6.

Since the 7 x 7 puzzle was lower in both graphs than the 6 x 6, there is a chance that our program could solve extremely large puzzles without a substantial amount of time. It is unlikely, but possible.

Conclusion

Our Futoshiki-solving program can be labeled as a success, as it solved each of our test puzzles quickly and correctly. From looking up Futoshiki puzzles online, it appears that most puzzles are 5 x 5 or 6 x 6, which our program seems to be able to handle, so our program should be quite useful. The main purpose of our program is to help players get the solution of difficult Futoshiki puzzles that they have given up on, and we believe that our program can indeed do that.

Even though we did not test our program on very large puzzles such as 15 x 15, we think it is safe to assume that our program can solve them. This is because our program checks for every possibility that satisfies the constraints and every Futoshiki puzzle has a unique solution, so it is impossible to fail unless we have a bug. Since our program easily solved five different puzzles of varying sizes, it also unlikely that we have any bugs.

Logic puzzles are very commonly played around the world. The point is to try to figure it out and solve it yourself, but there are times when you concede and yearn for the solution. When the solution cannot be found, programs that solve logic puzzles are the answer. If Futoshiki puzzles can be solved by a program made with basic CSP programming, then programs that solve any type of logic puzzle can also be created.

References

[1] http://www.theguardian.com/world/2006/sep/30/japan.estheraddley