CS360 Project 2 – Solving Strimko by Resolution

1 Introduction

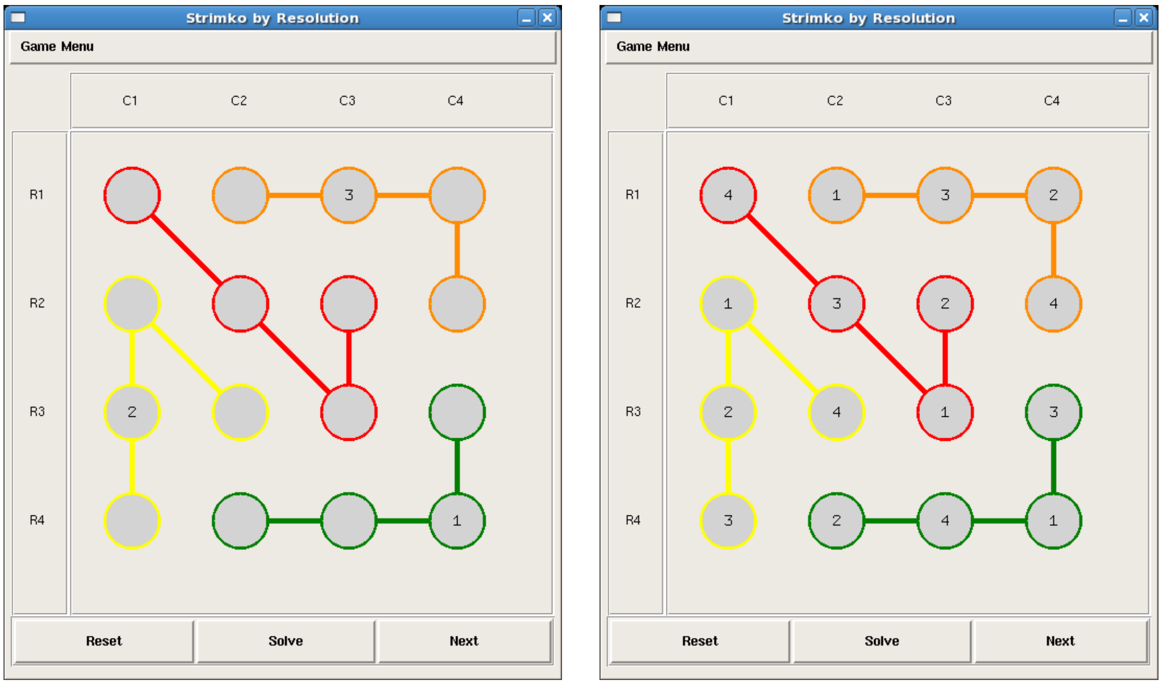

Figure 1: Left: A Strimko puzzle, initial board. Right: Solved (final) board.

Strimko is a popular puzzle on an $n \times n$ board, where apart from the n rows and columns, there are n streams/chains each of length n, as shown in varying colors in Figure 1. Each of the n^2 cells must be filled with a natural number $\in [1, n]$. The constraint is that no number can appear more than once in any row, column, or stream. Some of the cells are initially filled (Figure 1(left)), and you would want to fill the remaining cells without violating the constraint. Each board has a unique solution (e.g., Figure 1(right)) that can be reached by logical inference alone.

2 Knowledge Representation

The solver's knowledge of the puzzle's rules will be represented in propositional logic. We have two choices:

- Encode the complete knowledge base upfront, and then only make inferences about cell content without further expanding the knowledge base.

- Start with fewer facts, but with each new inference expand the knowledge base to enable further inferences.

We will take the second (incremental) approach.

Consider an $n \times n$ Strimko board. So, there are n rows, n columns, and n streams/chains of length n each. Suppose c_{ij}^k represents the proposition that cell (i, j) contains number k. If we reach the conclusion that (i, j) cannot contain k, it is represented as $\neg c_{ij}^k$. Then, for each row i and for each number k, we will have a disjunction of the form:

$$c_{i1}^k \vee c_{i2}^k \vee \cdots \vee c_{in}^k$$

This disjunction says that number k must be somewhere in row i. For $k \in [1, n]$, this implies that no number can appear more than once in a row. Since $k \in [1, n]$, there will be n disjunctions of this form for each row, and n^2 disjunctions over all rows. There will be a similar disjunction for each column and each chain as well, of which there are n each. Therefore, there will be a total of $3n^2$ disjunctions of this form.

Now for each cell (i, j) we will also have a disjunction of the form:

$$c_{ij}^1 \vee c_{ij}^2 \vee \cdots \vee c_{ij}^n$$

which says that each cell must be filled with some number $\in [1, n]$. Again there will be n^2 disjunctions of this form, over all cells.

The above $4n^2$ disjunctions constitute the base set of facts, that will be encoded before considering the (accumulative) evidence. For each evidence $c_{ij}^k = True$, i.e., when we know (i, j) must contain k for sure, we can add to the knowledge base the following conjunction for row i:

$$\neg c_{i,1}^k \wedge \neg c_{i,2}^k \wedge \cdots \wedge \neg c_{i,j-1}^k \wedge \neg c_{i,j+1}^k \wedge \cdots \wedge \neg c_{i,n}^k$$

This conjunction says that number k can only occur in one cell in the entire row i. Such a conclusion can also be made for the same evidence for column j as well as its chain. Moreover, the same evidence will lead to the following conjunction:

$$\neg c_{i,j}^1 \wedge \neg c_{i,j}^2 \wedge \cdots \wedge \neg c_{i,j}^{k-1} \wedge \neg c_{i,j}^{k+1} \wedge \cdots \wedge \neg c_{i,j}^n$$

which says that cell (i, j) may contain nothing other than k. Thus there will be 4 conjunctions of length $n - 1$ each as shown above, for each evidence. These can be broken into $4(n - 1)$ independent propositions that are all true. These new facts will expand the knowledge base.

3 Problem 1 – Resolve Strimko: 13 points

(**Part A: 10 points**) Implement the algorithm given in Algorithm 1. You need to complete the function `resolveStrimko()` in `proj2.cpp`. The inputs to the algorithm are sets A and B. Suppose we have p evidences initially (input B), i.e., $4p(n - 1)$ initial facts as described in the previous section. Each of these will reduce the length

Algorithm 1 resolveStrimko(A, B)

Input: Set A of current disjunctions, set B of p current facts.
Output: Potentially updated sets A and B, plus a flag indicating solve status
Local: Two sets *new* and *old*, latter initialized to B
while $|B| < n^2$ **do**
  $new \leftarrow \emptyset$
  for all $c_{ij}^k \in old$ **do**
    $C \leftarrow$ Generate a set of (up to $4(n-1)$) negative facts for empty cells in row i, column j and chain of (i, j)
    for all disjunction $d \in A$ **do**
      for all $\neg x \in C$ **do**
        if d contains x **then**
          Remove x from d
        end if
      end for
      if $|d| = 0$ **then**
        Return (A, B, "Contradiction")
      end if
    end for
    for all disjunction $d \in A$ **do**
      if $|d| = 1$ **then**
        Remove d from A and add it to *new* (unless d is already in B)
      end if
    end for
  end for
  if *new* is empty **then**
    Return (A, B, "Unsolved")
  end if
  Add *new* to B
  $old \leftarrow new$
end while
Return (A, B, "Solved")

of one (of the $4n^2$) disjunctions (input A) by 1, according to *disjunctive syllogism* (a special form of *resolution*) that takes the following form:

$$\frac{x_1 \vee x_2 \vee \cdots \vee \ x_n, \neg x_1}{x_2 \vee x_3 \vee \cdots \vee \ x_n}$$

After each of the $4p(n-1)$ facts have made their reductions on the disjunctions, if there is a disjunction with length 0, then there must have been a contradiction in A and B – *resolution refutation*. Note that if B is simply the initial set of facts, then this cannot happen. If there is any disjunction with length 1 (i.e., a conclusion of the form $c_{ij}^k = True$) then this may be a newly inferred cell or may have been already inferred before. (**Part B: 3 points**) Give an example showing that it may have been inferred before. If new, then in its turn it may produce **up to** $4(n-1)$ new facts, and the inference procedure continues this way. But if neither the input has been refuted, nor is there any disjunction of length 1 (i.e., a newly inferred cell), then `resolveStrimko()` cannot solve this puzzle, and returns status "Unsolved".

4 Problem 2 – Solve Strimko: 17 points

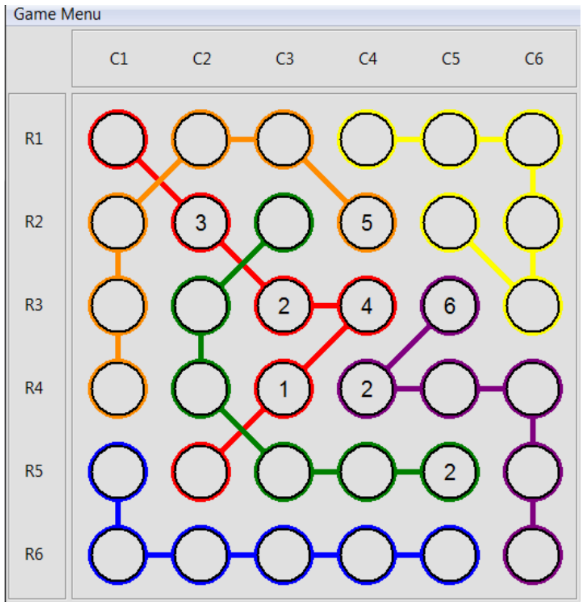

Figure 2: Left: An unsolved but solvable board.

The inference procedure in Algorithm 1 is sound because it is based on disjunctive syllogism which is sound. However, it is not complete. Figure 2 shows a puzzle where it can make no further inference, but you can. For instance, considering the red stream, you know that $(C2, R5)$ must contain either 5 or 6. But if it did contain 5, then the green stream couldnt contain 5 anywhere – a contradiction. Therefore $(C2, R5)$ must contain 6. This line of reasoning is *depth-first (backtracking) search* (DFS).

(**Part A: 10 points**) Implement the function `solveStrimko()` in `proj2.cpp`. This function is a DFS function that will call `resolveStrimko()` as a subroutine, and use its output to make a *guess* and continue until a contradiction is reached. Now you should be able to solve all included puzzles, even the ones marked "Unsolved" in Problem 1.

(**Part B: 5 points**) Prove that `solveStrimko()` is complete.

(**Part C: 2 points**) Prove that Sudoku[1] (a more popular puzzle) is actually a special case of Strimko. That is, `solveStrimko()` can be applied to solve Sudoku as well.

4.1 Testing and Visualizing

The program `proj2.cpp` takes 1 argument `problem_number`, which can be either 1 or 2. You can also run the program on one specified puzzle with `your_program_name problem_number size_id`.

[1]http://en.wikipedia.org/wiki/Sudoku

The input of both functions is an instance of class `Puzzle`, which contains `disjunction`, `evidence`, and `status`[2]. The functions return a `Puzzle` instance, with all inferred evidences in `evidence` and whether it's solved or not in `status`. Each element in `disjunction` and `evidence` is an instance of struct `Cell`, which denotes $c_{i,j}^k$.

As in Project 1, we provide a python script[3] to visualize the Strimko problem and your solutions.

We have 23 different puzzles of size $\in [4, 7]$. If your implementation is correct, an output file named `proj2_size_id.out` will be generated for each puzzle. Both the input and output files are in the `puzzles` folder. Check that the output files indeed exist. Then, you can visualize your solution by executing `python GUI.py`.

Note: The output files are overwritten when you execute the code again. Be careful and do not forget to backup your files.

5 Submission

You should submit one archive file named `YourName_proj2.zip`. This file should include only one code file (`proj2.cpp`) along with a PDF document file (`yourname_proj2.pdf`).

In `proj2.cpp`, you should complete the two functions `resolveStrimko()` and `solveStrimko()`. Put your name and student ID as comments in the code file.

In `yourname_proj2.pdf`, you should 1) describe which compiler/OS you use, 2) provide the results (screenshots) of all 5 6×6 puzzles for both algorithms, and 3) write down your answers to problems 1A, 2B, and 2C.

You should submit your archive file through the blackboard system by **Wednesday, March 25, 11:59pm**.

We have created a discussion board for this project on the blackboard system. Please also feel free to ask the TA in case you have any questions about the project.

[2](`disjunction`, `evidence`, `status`) is the same as (A, B, status) in Algorithm 1

[3]Modified from http://modelai.gettysburg.edu/2014/strimko/strimko_basecode.zip