My solution uses Common Lisp, specifically the SBCL implementation that is available here: http://www.sbcl.org/. On this website you can obtain the source that can be built as described in the install file provided with the source code. SBCL can also be installed on certain linux machines using the package systems (ubuntu: sudo apt-get install sbcl).

Once installed, the program can be run by issuing the following command: ./script.lisp inputfile, which loads the given input file and will attempt to solve the given board. If a solution was found, the peg numbers described by the jumps are given using the new board as a base. So if the first jump moves peg 1 onto peg 4, then in the next move the previous peg 2 is now peg 1.

Input format is based on files provided here: http://classes.soe.ucsc.edu/cmps101/Winter13/hw/prog3/Inputs/

The program works well for input sizes below 20 pegs, but once that point is reached most boards take longer and longer to compute. The problem stems mostly from the amount of enumerations that a given board possibly has. I tested the code using profiling on small boards to make sure the algorithm worked as expected and my assumptions held. But once you get into the larger boards the numbe of possibilities explodes.

I would think as long as the board is solvable it should not take too many iterations to break the board down into a problem that is already solved, but this didn't seem to be the case. The program still seemed to go over a lot more possiblities than I think would be needed and I was unable to find the reason why. Using profiling it's clear that the hash table helps as the call log shows that the solved? branch of the cond in solve-board is called at most half as many times as solve-board is called. This gap grows as the size of the board grows. For example, here is the profiling report for a board of 18 single pegs:

seconds	gc	consed	calls	sec/call	name
2.502	0.000	0	20,998	0.000119	HASH-FIND
0.262	0.000	444,096	1,812	0.000145	HASH-INSERT
0.012	0.000	2,359,296	10,498	0.000001	JUMP-PEG
0.011	0.000	2,359,296	10,498	0.000001	CLEAN-BOARD
0.006	0.000	458,704	23,484	0.000000	PEG-CAN-JUMP
0.001	0.000	196,608	1,814	0.000001	SOLVED?
0.000	0.000	0	1	0.000000	SOLVE-PUZZLE
0.000	0.000	0	1	0.000000	PRINT-SOLUTION
0.000	0.000	2,392,064	10,499	0.000000	SOLVE-BOARD
2.794	0.000	8,210,064	79,605		Total

20 pegs seems to be a magic number for some reason as the program takes exponentially longer time to solve and I wasn't able to find out why. In the previous result, the hash-find function takes .000119 seconds per call, but with 20 pegs that time goes up to .11. It's possible that the use of a generic list is what's causing the problem, but I didn't have enough time to switch the implementation once I got the initial program running. I think with more time that would probably be the first area I would check.