CSCI 5654

Linear Programming

Sudoku Solver- A binary integer linear programming model for Sudoku

Narain Yegneswara Sharma
Ranganathan Chidambaranathan
Santhanakrishnan Ramani

1. Introduction

Sudoku is a mathematical and logical puzzle with a grid containing nine large blocks. The nine large blocks in turn have a 3*3 grid each. Each puzzle appears with some clues ie some cells have been filled at the beginning. The aim is to fill the rest of the 9*9 grid such that all the subgrids contain numbers from 1-9 once. There are a few other rules or constraints that must be followed while filling the grids. The 9*9 grid containing 3*3 grid is by far the most commonly played puzzle. But, there are variations to this puzzle. Puzzles with 4*4 grid, containing 2*2 subgrids are also played.

This report is organized as follows. Section 2 shows the formulation of a sudoku puzzle as an Integer Linear Programming Problem. Section 3 talks about solving the ILP using branch and bound heuristic. In Section 4, we talk about using cutting plane method to solve the ILP. Section 5 shows the results and conclusion.

2. Sudoku as a Integer Linear Programming Problem:

Sudoku can be modelled as a Integer Linear Programming problem. In this project, we have modelled a sudoku puzzle as a Binary Integer Linear Program for solving a n*n grid puzzle.

Let's take n = 9. Each cell of the puzzle is divided into 9 cells, each cell can be either a 0 or 1. Since there are 81 such cells, we get 729 cells totally. These are the decision variables for the problem. Thus we have 729 decision variables.

Decision Variables:

Formally defining the decision variables,

Xijk = 1, if element (i, j) of the n × n Sudoku matrix contains the integer k
= 0, otherwise.

Once we get the values of all the decision variables, we know the number k that is present in each cell.

Constraints:

The knowledge of sudoku rules need to be encoded as the constraints for the linear programming problem. There are five rules primarily, and these have been converted as equality constraints.

1) In each row i of the matrix, there is exactly one value out of the digits from 1 to 9.

$$\sum_{i=1}^{n} A(i,j,k) = 1$$

2) In each column j of the matrix, there is exactly one value out of the digits from 1 to 9.

$$\sum_{j=1}^{n} A(i,j,k) = 1$$

3) In each depth k of the matrix, there is exactly one value out of the digits from 1 to 9.

$$\sum_{k=1}^{n} A(i,j,k) = 1$$

4) In each subgrid (0,3,9) of the matrix, there is exactly one value out of the digits from 1-9.

$$\sum_{i=1}^{3}\sum_{j=1}^{3} A(i+X, j+Y, k) = 1$$

where X, Y $\in \{0,3,6\}$

5) The clue matrix can be included as constraints in the Aeq and beq matrix. This is because those particular indexes are already provided and must not be changed.

$$A_{ijk} = 1 \; \forall (i,j,k) \; in \; clueMatrix$$

This can also be done by setting the lower bound of those particular indices as 1. This restricts the values in those indices from being anything else other than 1 since the upper bound is also set as 1.

Objective Function:

The sudoku problem is a type of constraint satisfaction problem. It is enough if the constraints are satisfied and we end up with a feasible solution. There is nothing to be optimized as such. Hence an objective function is not needed in these sort of problems. However, we have a non-constant objective function which helps in speeding up the solver.

$$f = x_1 + x_2 + x_3 + ... + x_n$$

n= 1,2,3 …. 729.

L1 - Norm Minimization

The solution of BILP model of sudoku gives a sparsest solution, which can be solved by using L0 Norm minimization. However, L0 norm minimization is considered to be NP-Hard. Hence we use the much simpler L1 Norm minimization to get the solution of a sudoku puzzle.

We formulate the LP as follows:

$$Min||X||_1 \ st. \ Ax = b$$

$$X = \sum_{i=1}^{729} x_i$$

The L1 norm minimization of the LP gives us the sparsest ‘x’ which solves the sudoku.

3. Branch and Bound

After Defining the rules of Sudoku, instead of calling the intlinprog function, our BranchandBound function is called which solves the Integer linear program same as the intlingprog function.

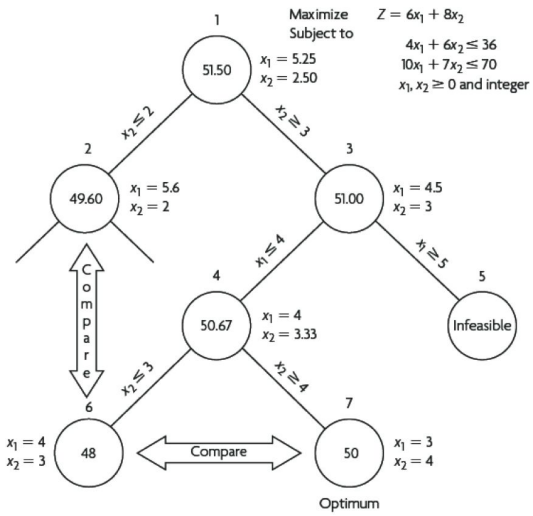
Divide and conquer method is the main strategy in our Branch and Bound Technique. Our main problem is divided into smaller parts, which is also called as Branching. After Branching, each parts optimal solution is computed to know how good a solution we can get for each smaller parts but to perform this we have to introduce a Bound Variable and branch the problem, until we get a better solution. For computing the optimal solution we are performing the linear programming relaxation of the integer programming

Consider a Model P, which is an integer program, the LP model that is obtained by not considering the requirement that all variables must be integers, is an LP relaxation of our model P.
Below are the steps involved in the Branch and Bound Method:

1. Divide a problem into sub problems also know as Branching.
2. Calculate the LP relaxation of the sub problem
   If (The LP problem has no feasible solution):
   Stop
   Else If (The LP problem has an integer optimal solution):
   Stop
3. Now start Comparing the optimal solution value with the best solution, which is already obtained.
   If (The LP problem has an optimal solution value that is smaller than the already existing solution):
   Stop

From the above cases we can come to a conclusion that we know about our sub problem and its optimal solution.

If (The LP problem has an optimal solution but not all of them are integers, and also the objective value is better than the already existing value):
Divide this sub problem and proceed further.
An Example of Branch and Bound Problem:



Code Implementation:

A function called BranchandBound is defined which is called by rulemodel function of Sudoku puzzle. BranchandBound(c,A,b,Aeq,beq,lb,ub,Integers) – This function calls does the LP Relaxation for the root node and further calls the Branch function which is recursive. The Branch Function further computes the LP relaxation of each node and performs the below steps:

1. **Checks for Infeasibility:**
   After computing the LP relaxation with the help of linprog , the EXITFLAG is checked if the problem is infeasible, if so then the node is pruned.
2. **Optimal Value computed with the help of Bound:**
   If the New Optimal value is better than the Bound, then the current objval is set as best objval.
3. **Branching:**
   If one of the optimal solution is not an integer then we branch on that variable further.
4. **Recurring/Repeat:**
   Until all the nodes are pruned and we get a best objval with optimal solution as integers we repeat these steps.

Cutting Plane

Introduction

The Cutting Plane method is one of the many existing popular and most widely used mathematical optimization method used for finding Integer solutions to mixed linear programming problems, as well as to solve general, not necessarily differentiable convex optimization problems by iteratively refining a feasible set or objective function by means of linear inequalities, termed cuts. The cuts are referred popularly as Gomory-Chvátal cuts as the cutting plane method to solve MILP was introduced by Ralph E. Gomory and Václav Chvátal.

Integer Linear Problem

Standard Form

Max $c^T x$

s.t. $Ax \leq b$
$x \geq 0$
$x \in \mathbb{Z}$

A, b, c are all assumed to be integers.

LP Relaxation

Max $c^T x$
s.t. $Ax \leq b$
$x \geq 0$

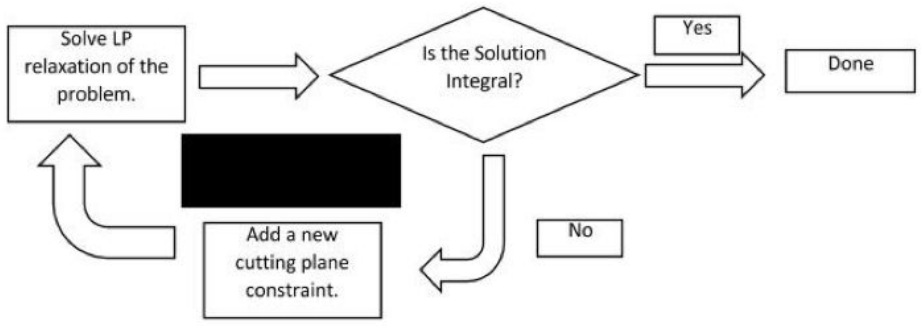
Conversion to Standard Form

- Equality constraints into two inequalities.
- Rewrite in case $x_i \geq 0$ is missing:

$$x_i \longrightarrow x_i^+ - x_i^-$$

- Convert $\geq$ to $\leq$ by negating both sides.

Algorithm of Cutting Plane



Cutting Plane Derivation

Randomly or by using some heuristics select a row from the final dictionary where the basic variable isn't an integer. Say $x_{Bk} = b_k + a_{k1}x_{I1} + .. + a_{kj}x_{Ij} + .. + a_{kn}x_{In}$

Cutting plane:

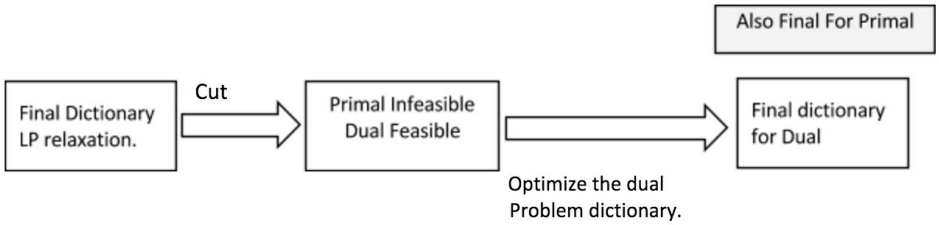
$$frac(-a_{k1})x_{I1} + .. + frac(-a_{kn})x_{In} \geq frac(b_k)$$

$$frac(x) = x - floor(x)$$

Cutting Plane after cut

- Resulting dictionary after adding cutting plane is primal infeasible, but the dual is feasible. So we proceed by solving the dual and convert it back to the primal once final dictionary is reached. The process is continued until integer solution is found.

'



Reconstructing the final dictionary after solving the LP Relaxation

Find the final set of basic and non-basic variables. Let set of basic variables be **bas**, and set of non-basic variables be **nonbas**

$$x_B = A_B^{-1}b \qquad - A_B^{-1}A_I x_I$$

$$c = c_B^T A_B^{-1}b \qquad + \ (-c_B^T A_B^{-1}A_I + c_I^T)\, x_I$$

*Reconstruction of **A matrix, b &c vector:***

A_b = A (:, bas);
A_i = A (:, nonbas);
c_b = c (bas, :)';
c_i = c (nonbas, :)';
pi = (A_b'\c_b')';
A = (A_b\A_i);
b = A_b\b;
c = (c_i - pi*A_i)';

GLPK

It is a set of routines written in the ANSI C programming language and organized in the form of a callable library. It is intended for solving linear programming (LP), mixed integer programming (MIP), and other related problems.

It supports the GNU MathProg modeling language, which is a subset of the AMPL language.

The GLPK package includes the following main components:

- primal and dual simplex methods
- primal-dual interior-point method
- branch-and-cut method
- translator for GNU MathProg
- application program interface (API)
- stand-alone LP/MIP solver

5. Results and Conclusion.

This project examined the possibility of formulating and solving the Sudoku puzzle using a Linear Programming approach. We looked at three possible approaches, using the in-built Intlinprog function in Matlab, branch and bound technique and finally using the cutting plane method. The Intlinprog and branch & bound method were successful in solving the Sudoku puzzle, whereas we had trouble with the cutting plane approach. After experimenting with the cutting plane method to solve Sudoku and general problems,we found out that this method is not efficient for solving Sudoku or 0-1 ILPs in general ,as the probability of the matrix Ab that needs to be found out during the process of simplex, being singular, is extremely high.

We have done a comparative study on the efficiency of each method, measuring the time taken for each method to compute the answer. We calculated two times, one that of either Intlinprog or linprog for returning an answer and the second of the entire running time of the program.

It is observed that the Intlinprog is much faster than the branch and bound technique. We changed various options in Intlinprog, such as using various levels of cuts, and various heuristics for choosing the branching in branch and bound and observed that choosing ‘maxpscost’(pesudo-cost) as the branching heuristic provided the fastest solution of 0.01 s.

In a few cases, we observed that there were more than 1 unique solution for a puzzle. In this project we have not focussed on finding all the solutions. We found that the method used for solving the LP played a role in selecting the solution.

Table 1. compares the running time of various methods involved and the heuristics tried.

		9 * 9 Sudoku		4 * 4 Sudoku	
Method	Heuristic	Computing X	Overall Running time	For computing X	Overall Running time
IntlinProg (Branch and Bound)	Most fractional	0.03	0.25	0.02	0.06
	Maxpscost	**0.01**	**0.25**	**0.01**	**0.05**
	Maxfun	0.03	0.25	0.01	0.07
Cuts	Basic	0.03	0.26	0.01	0.23
	Intermediate	0.04	0.27	0.01	0.19
	Advanced	0.04	0.29	0.02	0.20
Branch and Bound (Custom code)	-	0.11	0.36	0.08	0.26
GLPK	-	0.02	0.34	0.02	0.47

Table 1.

Table 2. shows the number of puzzles solved by each method.

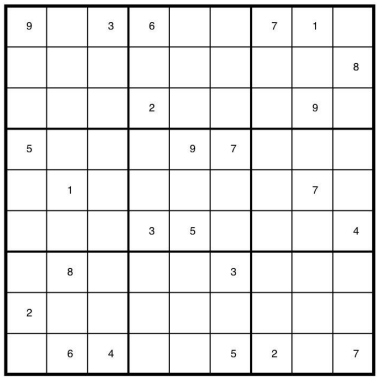
Type of Puzzle	Total Number of puzzles	Number of puzzles Intlinprog Solved	Number of puzzles Branch and Bound Solved
9 * 9	10	10	8
4 * 4	5	5	5

Table 2.

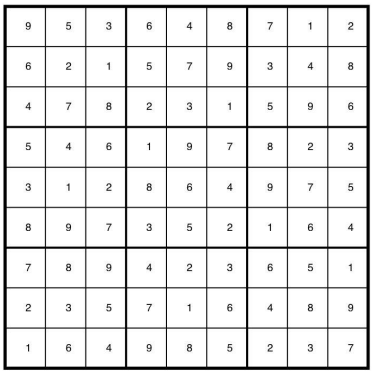
Output:

The following figure shows an “evil” level puzzle from website, websudoku.com. The clue is represented on the left followed by the solved puzzle on the right.

Inital Puzzle



Solved Puzzle



References:

1) Prabhu Babu, Kristiaan Pelckmans, Petre Stoica and Jian Li, Linear Systems, Sparse Solutions, and Sudoku, IEEE Signal Processing Letters, Vol. 17, No. 1, Jan 2010.
2) Andrew C. Bartlett, Timothy P. Chartier, Amy N. Langville and Timothy D. Rankin, An Integer Programming Model for the Sudoku Problem.
3) Website. [Online]. Available: http://www.websudoku.com/
4) Website. [Online]. Available: https://www.mathworks.com/help/optim/ug/solve-sudoku-puzzles-via-integer-programming.html
5) Website. [Online]. Available: https://www.mathworks.com/help/optim/ug/intlinprog.html
6) Website. [Online]. Available: https://rorasa.wordpress.com/2012/05/13/l0-norm-l1-norm-l2-norm-l-infinity-norm/
7) Website. [Online]. Available: https://en.wikipedia.org/wiki/Cutting-plane_method
8) Website. [Online]. Available: https://www.gnu.org/software/glpk/
9) Website. [Online]. Available: http://www4.ncsu.edu/~kksivara/ma505/codes/MILP_bb.m
10) Website. [Online]. Available: http://www4.ncsu.edu/~kksivara/ma505/codes/MILP.m
11) Textbook. McGraw-Hill/Irwin 7S–15 (Figure 7S-10)

12) Classroom Slides.