University of Konstanz
Department of Mathematics and Statistics
Summer Term 2012

Sebastian Gruler
María López Quijorna
Markus Schweighofer

Polynomial Optimization – Computer Project 3

In this project you create a MATLAB function `function G = sudrelax(N,P)...` that *tries* to solve a generalized sudoku puzzle P of size $N^2 \times N^2$. For $N \in \mathbb{N}$, an *N-sudoku* is a matrix $S \in \{1, \dots, N^2\}^{N^2 \times N^2}$ such that in each row, each column and each block of S the numbers 1 to N^2 appear exactly once. The term *block* refers here to the fact that S can be written as an $N \times N$ matrix of $N \times N$ blocks.

An *N-sudoku puzzle* is a matrix $\{0, 1, \dots, N^2\}^{N^2 \times N^2}$ where the zeros can be *uniquely* replaced to become an N-sudoku. The aim of the sudoku solver is to return, upon input of $N \in \mathbb{N}$ and an N-sudoku puzzle P, an N-sudoku S such that S and P agree at all non-zero entries of P.

2-sudokus are also called *shidokus* and 3-sudokus are simply called *sudokus*. Your sudoku solver should be able to solve most shidokus and, if possible, also easy sudokus.

You should not expect to much from your sudoku solver since you are not allowed to do any kind of sophisticated trial and error or backtracking procedures. Formally, we could say that your algorithm must be a polynomial time algorithm, i.e., there must be a polynomial $p \in \mathbb{R}[T]$ such that your algorithm terminates its search for each N-sudoku after at most $p(N)$ instructions (but don't care about this formalism, just avoid trial and error). Moreover, you are allowed to call a linear or semidefinite programming solver only *once* per puzzle. Your algorithm must return only a *guess* $G \in \mathbb{R}^{N^2 \times N^2}$ for the right solution. As said above, this guess should be correct in easy cases. You can test your algorithm using sudoku generators on the internet or for MATLAB[1] (for the latter, try `[sol,sud]=suma(1,0,0,323,0)`).

If you use moment relaxations, you are by now allowed to use the built-in moment[2] and sums-of-squares[3] modules of YALMIP. We encourage you to experiment with moment relaxations but we warn you that there might be much better approaches.

All files have to contain a comment with your name in the first line. The files should include comments in English language that facilitate the understanding of your approach.

Hint: It is possible to use an LP relaxation (in this case use an LP solver instead of an SDP solver!) based on a decision variable `D=sdpvar(N^2,N^2,N^2,'full')` constrained by `set(0<=D<=1)` (20 lines of MATLAB code can do the job but maybe you are more ambitious). Don't be disappointed if your relaxation does not work for hard sudokus[4] since there are good reasons for this.

Due by Thursday, July 12, 2012, 11:11 am.[5]

[1] `http://www.mathworks.com/matlabcentral/fileexchange/28168-sudoku-generator`
[2] `http://users.isy.liu.se/johanl/yalmip/pmwiki.php?n=Tutorials.MomentRelaxations`
[3] `http://users.isy.liu.se/johanl/yalmip/pmwiki.php?n=Tutorials.SumOfSquares`
[4] `http://en.wikipedia.org/wiki/Sudoku_algorithms#Exceptionally_difficult_Sudokus_.28hardest_Sudokus.29`
[5] The best solution will be rewarded by a glass of wheat beer in the university beer garden.