Entropy Minimization for Solving Sudoku

Article *in* IEEE Transactions on Signal Processing · January 2012

DOI: 10.1109/TSP.2011.2169253 · Source: DBLP

2 authors:

Jake Gunther
Utah State University
196 PUBLICATIONS **1,110** CITATIONS
SEE PROFILE

T.K. Moon
Utah State University
214 PUBLICATIONS **5,814** CITATIONS
SEE PROFILE

Entropy Minimization for Solving Sudoku

Jake Gunther *Senior Member, IEEE* and Todd Moon *Senior Member, IEEE*

Abstract

Solving Sudoku puzzles is formulated as an optimization problem over a set of probabilities. The constraints for a given puzzle translate into a convex polyhedral feasible set for the probabilities. The solution to the puzzle lies at an extremal point of the polyhedron where the probabilities are either zero or one and the entropy is zero. Because the entropy is positive at all other feasible points, an entropy minimization approach is adopted to solve Sudoku. To escape local entropy minima at non-solution extremal points, a search procedure is proposed in which each iteration involves solving a simple convex optimization problem. This approach is evaluated on thousands of puzzles spanning four levels of difficulty from "easy" to "evil".

Index Terms

Convex optimization, Sudoku

I. Introduction

Sudoku is a puzzle in which $N = n^2$ different symbols (usually digits 1 through N) are to be arranged in an $N \times N$ array such that the arrangement agrees with given clues and meets the puzzle constraints. Solving Sudoku puzzles is of interest in the signal processing community because it has ties to decoding error correcting codes [1]–[3]. Yato and Seta are generally attributed as showing Sudoku to be an NP-complete problem [4], [5].

Approaches to solve Sudoku can be divided into two groups. In the first group are techniques guaranteed to find one or all solutions if they exist. Because of NP-completeness, all known approaches in this category have a complexity that increases exponentially in the size of the puzzle. Backtracking, covering

The authors are with the Deptartment of Electrical & Computer Engineering at Utah State University, 4120 Old Main Hill, Logan, UT 84322-4120. E-mail: jake.gunther@usu.edu, todd.moon@usu.edu.

and brute force approaches are described in [6] and constraint programming in [7]. Exact covering implemented via Knuth's Dancing Links algorithm [8] is probably the most efficient method for solving Sudoku. In [9] Sudoku puzzle constraints were cast as system of linear equality constraints along with binary integer constraints on the parameters, and puzzles were solved using a binary integer linear program solver.

The second category of Sudoku "solvers" do not guarantee an exact solution that meets all problem constraints. Instead, this category of algorithms seeks to find near-optimal approximate "solutions", and in many instances the optimum is attained. Sacrificing guarantees of optimality may be viewed as acceptable for several reasons. Suboptimal algorithms can offer lower complexity than exact solution methods. Because suboptimal algorithms are often derived by relaxing some of the hard problem constraints, they can be applied to other related problems where exact solution methods are not applicable. Such is the case with the method presented in this paper as shown in the sections that follow.

We note that the Dancing Links algorithm [8] solves any 9×9 puzzle nearly instantaneously on a modern computer. No other exact or approximate solver of which we are aware can compete with Dancing Links for speed. We emphasize that our long-range interest is not in Sudoku *per se* but rather in applying what is learned from Sudoku to related problems such as error correction decoding, where information about bits in a codeword are often available in the form of prior probabilities rather than as hard bit decisions. Though these applications are beyond the scope of this paper, we show that the proposed technique can be applied to stochastic Sudoku puzzles introduced in [1] where the clues are not be specified as specific digits but rather as probability distributions that given cells are filled by each of the digits. Dancing Links cannot be directly applied to solve stochastic puzzles or other problems where soft information is given because it operates on the basis of a hard objective, whether a condition is satisfied or not. The exact solver by Bartlett [9] can accommodate soft information, but as we show in Section V it can be slow even for 9×9 puzzles. By comparison, the method of this paper, as well as the one in [10], are fast and can also accommodate soft information.

There are a number of approximate methods for solving Sudoku including approaches based on simulated annealing [11] and quantum simulated annealing and genetic algorithms [12]. Geometric particle swarm optimization was shown to find near-optimal solutions [13], and Sinkhorn balancing was also shown to solve most puzzles to which it was applied [14].

What Bartlett [9] showed was that a solution to a 9×9 Sudoku puzzle can be represented as a 729×1 vector of 81 ones and 648 zeros that satisfied linear equality constraints. The sparsity of such a solution was exploited together with the linear constraints in [10] where Sudoku was solved as a l_1-norm

minimization problem. Thus, it was in [10] where Sudoku was first solved using traditional calculus-based optimization techniques.

In this paper, Sudoku is cast as the problem of minimizing a concave objective over a polyhedron defined by linear equality constraints (as in [9], [10]) and inequality constraints (because the parameters are interpreted as probabilities). This is a relaxation of the hard binary constraints used in [9] whereas no constraints were used in [10]. In the following, concave minimization over a bounded convex set is converted into a sequence of convex optimization problems which can be solved efficiently. While we have no proof that this technique will solve every puzzle, we show that it is capable of solving thousands of them with varying levels of difficulty, including puzzles rated as extremely difficult.

The remainder of the paper is organized as follows. Section II formulates Sudoku in terms of a vector of probabilities constrained to lie in a polyhedron and highlights connections to [9], [10]. Section III explains why entropy, a concave function, provides an effective objective to minimize for solving Sudoku puzzles. Section IV develops an optimization strategy and formulates an algorithm for minimizing a concave function over a bounded convex set, and Section V presents comparative results of different algorithms applied to thousands of puzzles.

II. Problem Formulation

For a 9×9 Sudoku puzzle with L given clues, Babu *et al* [10] represented the constraints (all of them linear) as $\mathbf{A}\mathbf{x} = \mathbf{1}$, where

$$\mathbf{A} = \begin{bmatrix} \mathbf{A}_s \\ \mathbf{A}_c \end{bmatrix}, \tag{1}$$

is a $(324 + L) \times 729$ matrix with nine 1's on each of the first 324 rows, one 1 on each of the last L rows, and is zero otherwise. The equation $\mathbf{A}_s\mathbf{x} = \mathbf{1}$ represents the 324 constraints satisfied by all 9×9 Sudoku puzzles, and $\mathbf{A}_c\mathbf{x} = \mathbf{1}$ represents the L clue constraints associated with the given puzzle.

The 729×1 vector $\mathbf{x}$ is a stack of 81 9×1 subvectors, one for each of the 81 cells in the puzzle. A solution $\mathbf{x}^\star$ to a given puzzle is characterized by each 9×1 subvector being zero except for a 1 in the position of the digit assigned to that cell. Therefore, a solution to a given Sudoku puzzle is any $\mathbf{x}^\star$ satisfying the two conditions

$$\mathbf{A}\mathbf{x}^\star = \mathbf{1}, \;\; \mathbf{x}^\star \in \{0, 1\}, \tag{2}$$

where $\mathbf{x}^\star \in \{0, 1\}$ is understood to apply to the elements of $\mathbf{x}^\star$. We assume that enough clues are given in any puzzle specification that the solution to (2) is unique.

In this paper, the elements of $\mathbf{x}$ are viewed as probabilities with $0 \leq \mathbf{x} \leq 1$, the inequality applying element wise. The element of $\mathbf{x}$ giving the probability of the i^{th} digit filling the $(j,k)^{\text{th}}$ cell in the puzzle is denoted x_{ijk}. A relaxation of the hard constraints in (2) which represents our model of $\mathbf{x}$ as probabilities is

$$\mathbf{A}\mathbf{x} = \mathbf{1}, \;\; \mathbf{x} \geq 0. \tag{3}$$

These constraints define a polyhedron which is referred to as the feasible set for a given puzzle. Note that any solution $\mathbf{x}^\star$ is also a feasible point. The advantage of the relaxation in (3) is two-fold. First, it is easy to find points in the feasible set. Second, it is easy to move along paths across the feasible set. This is the basis for our solution approach in Section III.

It is interesting to consider the size of the feasible set for a given puzzle. For some puzzles, the feasible set consists of a single point, i.e. the feasible set (3) and solution set (2) are equal. In other words, relaxing the binary constraint (2) to the non-negative constraint (3) is often sufficient to solve a puzzle. In these cases, the solution may be obtained by solving the constraint satisfaction problem

$$\text{find} \;\; \mathbf{x}, \;\; \text{subject to} \;\; \mathbf{A}\mathbf{x} = \mathbf{1}, \quad \mathbf{x} \geq 0, \tag{4}$$

which can be solved efficiently using standard software packages such as [15]. Puzzles having a single feasible point that is found by solving (4) are referred to as easy puzzles, otherwise puzzles will be referred to as difficult. In Section III difficult puzzles are subdivided into three levels of difficulty: medium, hard and evil.

For some puzzles, depending on the given set of clues, the feasible set defined by (3) contains an infinite number of points, but only one solution. In this case the solution to (4) provides a feasible point, not a solution, and an objective function is needed to judge the desirability of points in the feasible set.

Babu *et al* [10] recognized that the solutions characterized by (2) are sparse, i.e. only a few of the 729 elements of $\mathbf{x}$ are nonzero, and proposed that Sudoku puzzles could be solved by

$$\text{minimize} \;\; \|\mathbf{x}\|_0, \;\; \text{subject to} \;\; \mathbf{A}\mathbf{x} = \mathbf{1}. \tag{5}$$

In practice, they solved the relaxed l_1 problem

$$\text{minimize} \;\; \|\mathbf{x}\|_1, \;\; \text{subject to} \;\; \mathbf{A}\mathbf{x} = \mathbf{1}. \tag{6}$$

Notice that (6) omits the non-negativity constraint in (3). Were it to be included, the l_1 problem above would be equivalent to the feasibility problem in (4) because, for non-negative parameters, the 1-norm is constant ($\|\mathbf{x}\|_1 = \mathbf{1}^T\mathbf{x} = 81$) for all $\mathbf{x}$ satisfying $\mathbf{A}\mathbf{x} = \mathbf{1}$ and $\mathbf{x} \geq 0$. Thus, with the additional

constraint $\mathbf{x} \geq 0$, (6) is equivalent to (4) which is insufficient for solving difficult puzzles. Though (6) can be solved efficiently using standard software packages, it does not solve some puzzles. An example was given in [10] along with an iterative re-weighted 1-norm minimization algorithm that did solve the given puzzle.

In light of (2), Bartlett *et al* [9] cast the Sudoku problem as the binary integer constraint satisfaction problem,

$$\text{find } \mathbf{x}, \text{ subject to } \mathbf{A}\mathbf{x} = \mathbf{1}, \ \mathbf{x} \in \{0,1\}, \tag{7}$$

and solved this using Matlab's `bintprog` function which solves the optimization problem

$$\text{minimize } \mathbf{0}^T\mathbf{x} = 0, \text{ subject to } \mathbf{A}\mathbf{x} = \mathbf{1}, \ \mathbf{x} \in \{0,1\}. \tag{8}$$

The `bintprog` function applies a branch and bound tree search technique in which the tree is explored by linear programming (LP) subproblems, where the hard constraint in (2) is relaxed to $0 \leq x_{ijk} \leq 1$ for all but one of the remaining non-integer variables for which the hard constraint is imposed. Two LPs are solved: one in which the variable in question is constrained to be zero and the other in which the variable is constrained to be one. The tree search is discontinued along a branch if both LPs are found to be infeasible. The disadvantage of this approach is that it can be time consuming, especially for difficult puzzles.

III. Entropy for Sudoku

Although we use a similar representation of the constraints as in [9] and [10], our interpretation of $\mathbf{x}$ and our solution approach are very different. We interpret $\mathbf{x}$ as a set of probabilities subject to constraints. We visualize $\mathbf{x}$ as a point in the interior of the convex polyhedron defined by (3) and use efficient convex optimization routines to move across the polyhedron to an extremal point that solves the puzzle.

Define the 9×9 matrices formed from x_{ijk}, when one of i, j or k is held fixed, to be slices of the cube x_{ijk}: $\mathbf{X}_{i::}$ is an i-slice, $\mathbf{X}_{:j:}$ is a j-slice, and $\mathbf{X}_{::k}$ is a k-slice. Constraints (3) may be interpreted as saying that all slices of the cube x_{ijk} along any of its dimensions are doubly stochastic matrices, i.e. their rows and columns sum to one.

Suppose that $x^\star_{ijk}$ solves a given Sudoku puzzle. Then $x^\star_{ijk} \in \{0,1\}$ holds and the slices of $x^\star_{ijk}$ become permutation matrices, a special doubly stochastic matrix composed of 0's and 1's in which every row and column has exactly one 1. In addition, each 3×3 box contains exactly one 1. The location of the 1's in the i-slices give the locations of digit i in the solved puzzle. As an aside, we note that because

of the constraints, the slices are orthogonal under the trace inner product on matrices, i.e. the set of nine i-slices are pairwise orthogonal $\text{trace}(\mathbf{X}_{i_1::}^T \mathbf{X}_{i_2::}) = 0$ for all $i_2 \neq i_1$, and the same holds for the set of j-slices and the set of k-slices.

The Birkhoff-von Neumann theorem [16] states that every doubly stochastic matrix is a convex combination of permutation matrices. The Birkhoff polyhedron $\mathcal{B}_N$, which is the set of all $N \times N$ doubly stochastic matrices, is the convex hull of $N \times N$ permutation matrices. Put another way, the extreme points of $\mathcal{B}_N$ are permutation matrices.

Define $\mathcal{B}_9^9 = \mathcal{B}_9 \times \cdots \times \mathcal{B}_9$ (nine-fold Cartesian product of $\mathcal{B}_9$) as the convex hull of 9-tuples of 9×9 permutation matrices. The extreme points of $\mathcal{B}_9^9$, $\text{ext}(\mathcal{B}_9^9)$, are all possible 9-tuples of 9×9 permutation matrices. Define $\mathcal{S}_9 = \{\mathbf{x} | \mathbf{A}_s \mathbf{x} = \mathbf{1}, \mathbf{x} \geq 0\}$ to be the Sudoku (size 9) polyhedron, a convex set. It is a subset of $\mathcal{B}_9^9$. The extreme points of $\mathcal{S}_9$, given by $\text{ext}(\mathcal{S}_9) = \{\mathbf{x} | \mathbf{A}_s \mathbf{x} = \mathbf{1}, \mathbf{x} \in \{0,1\}\}$, are all possible solutions to all Sudoku puzzles. These extreme points are 9-tuples of orthogonal 9×9 permutation matrices in which each permutation matrix (the i-slices) satisfy 3×3 box constraints. The set $\mathcal{S}_9$ is the convex hull of these extreme points. The extreme points of $\mathcal{S}_9$ are also extreme points of $\mathcal{B}_9^9$.

A Sudoku puzzle specification gives additional clue constraints $\mathbf{A}_c \mathbf{x} = \mathbf{1}$ that a solution must satisfy. Let $\mathcal{P}_9 = \{\mathbf{x} | \mathbf{A}_s \mathbf{x} = \mathbf{1}, \mathbf{A}_c \mathbf{x} = \mathbf{1}, \mathbf{x} \geq 0\} = \{\mathbf{x} | \mathbf{A}\mathbf{x} = \mathbf{1}, \mathbf{x} \geq 0\}$ be the polyhedron corresponding to a given puzzle specification. The clue hyperplanes defined by $\mathbf{A}_c \mathbf{x} = \mathbf{1}$ cut across $\mathcal{S}_9$ and reduce it down to $\mathcal{P}_9$. If enough clues are given, only one extreme point of $\mathcal{S}_9$ remains in $\mathcal{P}_9$ and the puzzle is solved by finding this extreme point.

Summarizing ideas from above, the solution $\mathbf{x}^\star$ to a given Sudoku puzzle is an extremal point of $\mathcal{P}_9$ in which $\mathbf{x}^\star \in \{0,1\}$. But $\mathcal{P}_9$ may have extremal points $\mathbf{x}$ for which some elements of $\mathbf{x}$ are neither 0 nor 1. We propose to solve Sudoku puzzles by first finding a feasible point, say by solving (4), and then moving across the feasible set under the influence of a continuous objective function that favors points $\mathbf{x}$ that have 0 or 1 elements. By doing so, an extremal point of $\mathcal{P}_9$ is found which represents the solution to the puzzle.

The Shannon entropy [17] is a suitable objective function for solving Sudoku puzzles. For a vector $\mathbf{p} = [p_1, \cdots, p_n]$ satisfying $\mathbf{1}^T \mathbf{p} = 1$ and $\mathbf{p} \geq 0$, a probability mass function (PMF), the entropy is $H(\mathbf{p}) = -\sum_{i=1}^n p_i \log p_i$. The uniform distribution, where $p_i = 1/n$ for all i, yields a maximum entropy of $\log n$. A PMF where $p_i = 1$ for some i and $p_j = 0$ for $j \neq i$ yields a minimum entropy of zero.

For the problem at hand, x_{ijk} is a PMF when any two of its subscripts are fixed, e.g. fix i and j then

the entropy of $x_{ij:}$ is $H(x_{ij:}) = -\sum_{k=1}^{9} x_{ijk} \log x_{ijk}$. Summing over i and j yields the total entropy,

$$H(\mathbf{x}) = -\sum_{i,j,k} x_{ijk} \log x_{ijk}. \tag{9}$$

Feasible points in $\mathcal{P}_9$ that have probability distributed among many elements of $\mathbf{x}$ have higher entropy than the extreme point where the probability is concentrated in 81 of the elements ($\mathbf{x} \in \{0, 1\}$) for which the entropy is zero. The entropy function is concave. Thus, starting from any feasible point with high entropy and moving across the feasible set toward points of lower entropy should ultimately lead to the extreme point for $\mathcal{P}_9$ which solves the puzzle. Thus, an optimization problem for solving Sudoku puzzles is

$$\text{min. } H(\mathbf{x}), \quad \text{s.t. } \mathbf{A}\mathbf{x} = \mathbf{1}, \quad \mathbf{x} \geq 0. \tag{10}$$

Unfortunately this is not a convex optimization problem because it attempts to minimize a concave objective [18]. Maximizing in (10) is an easily solved problem (using software). The next section discusses ways to solve (10).

IV. Concave Minimization Over a Polyhedron

The minimum of entropy (a concave function) over a polyhedron is achieved at a vertex $\mathbf{x}^\star$ with local minima possibly occurring at other vertices. On any line between vertices, the entropy will be either monotonic or pass over a peak. Therefore, to find the global minimum of entropy on the feasible set $\mathcal{P}_9$, our algorithm alternates between stepping in directions that are locally up hill (to escape local minima) and locally down hill. We introduce the notation $\check{\mathbf{x}}$ for exploring entropy in descent directions and use $\hat{\mathbf{x}}$ when moving in directions of ascent.

We adopt a heuristic exploratory strategy for finding the minimum of entropy on the feasible set $\mathcal{P}_9$. Starting from a feasible point, a descent direction is derived from local information about the entropy. This direction is explored until a minimum of entropy is reached. If the minimum value is zero, corresponding to a $\{0, 1\}$ extremal point, then the puzzle is solved. If not, then the starting feasible point must be on the wrong side of the entropy peak. Therefore, a step is taken in a direction of increasing entropy with the objective of passing over to the opposite side of the peak. Then the descent exploration is repeated. Each iteration of this process involves solving convex optimization problems. As shown in Section V, this solves most Sudoku puzzles in a small number of iterations. A few remarks about each of these steps is provided and then a summary is given.

A. Finding a Feasible Point

Finding a point in the feasible set $\mathcal{P}_9$ for a given puzzle amounts to solving (4). Using convex optimization software, this problem is solved as

$$\text{minimize } \ 0, \ \text{ subject to } \ \mathbf{A}\mathbf{x} = \mathbf{1}, \ \ \mathbf{x} \geq 0. \tag{11}$$

Let $\hat{\mathbf{x}}$ be an initial feasible point. It is possible that $\hat{\mathbf{x}}$ solves the puzzle. This may be ascertained by checking $\mathbf{A}[\hat{\mathbf{x}}] = \mathbf{1}$, where $[\mathbf{x}]$ rounds the elements of $\mathbf{x}$ to zero or one: $[x] = 1$, if $0.5 \leq x \leq 1$, and $x = 0$ otherwise. If $[\hat{\mathbf{x}}]$ does not solve the puzzle, then additional steps are needed as described in the subsections that follow. The full initialization procedure may be characterized as follows.

Algorithm INIT: Find Initial Feasible Point

---

1) Solve the feasibility problem:

$$\text{minimize } \ 0, \ \text{ subject to } \ \mathbf{A}\hat{\mathbf{x}} = \mathbf{1}, \ \ \hat{\mathbf{x}} \geq 0.$$

2) If $\mathbf{A}[\hat{\mathbf{x}}] = \mathbf{1}$, then stop. The puzzle is solved.

B. Exploring Down Hill Directions

Using the first-order Taylor approximation for F around $\mathbf{x}$, $F(\mathbf{x}+\mathbf{z}) \approx F(\mathbf{x}) + \mathbf{y}^T\mathbf{z}$, where $\mathbf{y}$ is the gradient of F (assumed to be differentiable) at $\mathbf{x}$, the inner-product $\mathbf{y}^T\mathbf{z}$ may be interpreted as the change in F for small steps $\mathbf{z}$. The method of steepest descent for unconstrained minimization chooses a search direction $\mathbf{z}$ to maximize the decrease in F by minimizing $\mathbf{y}^T\mathbf{z}$ subject to $\|\mathbf{z}\| \leq 1$. Once the search direction is found, a line search is performed to minimize $F(\mathbf{x}+\mu\mathbf{z})$ with respect to μ.

Ideas from steepest descent may be applied to decrease entropy without leaving the feasible set $\mathcal{P}_9$. At a feasible point $\check{\mathbf{x}}$, let $\mathbf{y}$ be an ascent direction computed from derivatives of $H(\mathbf{x})$ at $\check{\mathbf{x}}$. Alternatives for $\mathbf{y}$ include the gradient

$$\mathbf{y} = \nabla H(\check{\mathbf{x}}) = \left.\frac{\partial H(\mathbf{x})}{\partial \mathbf{x}}\right|_{\mathbf{x}=\check{\mathbf{x}}} = -\log\check{\mathbf{x}} - \mathbf{1} \tag{12}$$

and the Newton direction

$$\mathbf{y} = -\left[\frac{\partial^2 H(\mathbf{x})}{\partial \mathbf{x}\partial \mathbf{x}^T}\right]^{-1} \left.\frac{\partial H(\mathbf{x})}{\partial \mathbf{x}}\right|_{\mathbf{x}=\check{\mathbf{x}}} = -\check{\mathbf{x}} \odot \log\check{\mathbf{x}} - \check{\mathbf{x}}, \tag{13}$$

where $\odot$ stands for element-wise product of vectors and the logarithm is applied element wise to $\mathbf{x}$. To decrease H, $\check{\mathbf{x}}$ should be moved in a direction opposite to $\mathbf{y}$. However, it is conceivable that $\check{\mathbf{x}} - \mu\mathbf{y}$ is

infeasible for all $\mu > 0$. A search direction that keeps the updated point $\check{\mathbf{x}} + \mathbf{z}$ in $\mathcal{P}_9$ can be found by solving the following convex optimization problem:

$$\text{min. } f(\mathbf{z}), \text{ s.t. } \mathbf{A}(\check{\mathbf{x}} + \mathbf{z}) = \mathbf{1}, \ \check{\mathbf{x}} + \mathbf{z} \geq 0, \tag{14}$$

where $f(\mathbf{z}) = \mathbf{y}^T\mathbf{z}$ or $f(\mathbf{z}) = \|\alpha\mathbf{y} + \mathbf{z}\|$. Since $\mathbf{y}$ points in a direction of increasing entropy, when $f(\mathbf{z}) = \mathbf{y}^T\mathbf{z}$ (14) looks for $\mathbf{z}$ that points as opposite to $\mathbf{y}$ as possible subject to the condition that taking the step $\check{\mathbf{x}} + \mathbf{z}$ remains in the feasible set for the puzzle $\mathcal{P}_9$. The objective $f(\mathbf{z}) = \|\alpha\mathbf{y} + \mathbf{z}\|$ measures directions opposite to $\mathbf{y}$ using a norm, which is minimized when $\mathbf{z}$ is as close to $-\alpha\mathbf{y}$ as possible. The scalar α is a user defined parameter used to accelerate convergence. Note that when using the norm objective in (14), the solution $\mathbf{z}$ is not the projection of $-\alpha\mathbf{y}$ onto the feasible set because the update $\check{\mathbf{x}}+\mathbf{z}$ is also constrained to be feasible. While any valid norm may be considered, the results in Section V considered only the l_1 and l_2 norms.

After each descent step the test point $\check{\mathbf{x}}$ is replaced by $\check{\mathbf{x}} + \mathbf{z}$, $\check{\mathbf{x}} \leftarrow \check{\mathbf{x}} + \mathbf{z}$. The termination condition $\mathbf{A}[\check{\mathbf{x}}] = \mathbf{1}$ is applied to the updated $\check{\mathbf{x}}$. The descent procedure is started from the most recent up hill step: $\check{\mathbf{x}} = \hat{\mathbf{x}}$. The full descent procedure may be characterized as follows.

Algorithm DOWN: Entropy Descent

1) Compute an ascent direction $\mathbf{y}$ at $\check{\mathbf{x}}$ using (12) or (13).
2) Find a feasible step $\mathbf{z}$ by solving

$$\text{minimize } \mathbf{y}^T\mathbf{z} \text{ or } \|\alpha\mathbf{y} + \mathbf{z}\|,$$

$$\text{subject to } \mathbf{A}(\check{\mathbf{x}} + \mathbf{z}) = \mathbf{1}, \ \check{\mathbf{x}} + \mathbf{z} \geq 0.$$

3) Update: $\check{\mathbf{x}} \leftarrow \check{\mathbf{x}} + \mathbf{z}$.
4) If $\mathbf{A}[\check{\mathbf{x}}] = \mathbf{1}$, then stop. The puzzle is solved.

C. Stepping Over the Entropy Peak

Descending entropy from a starting feasible point $\check{\mathbf{x}} \in \mathcal{P}_9$ by applying multiple descent steps may not terminate in a solution point with zero entropy which solves the puzzle. This condition is easily detected by testing, for example, the norm of the update $\mathbf{z}$, which is zero when no further descent steps can be taken. In this case, the starting feasible point $\hat{\mathbf{x}}$ was on the wrong side of the entropy peak. A new descent search may begin starting from a feasible point on the opposite side of the peak.

Points on the opposite side of the entropy peak may be reached by stepping in a direction of locally increasing entropy. Such steps can be found by replacing the minimization in (14) by maximization. This

in fact steps past the entropy peak to the opposite side of the feasible set in the direction of locally fastest increasing entropy. The full ascent procedure may be characterized as follows.

Algorithm UP: Entropy Ascent

1) Compute an ascent direction $\mathbf{y}$ at $\hat{\mathbf{x}}$ using (12) or (13).
2) Find a feasible step $\mathbf{z}$ by solving

$$\text{minimize} \quad -\mathbf{y}^T\mathbf{z} \;\text{ or }\; \|\alpha\mathbf{y}-\mathbf{z}\|,$$

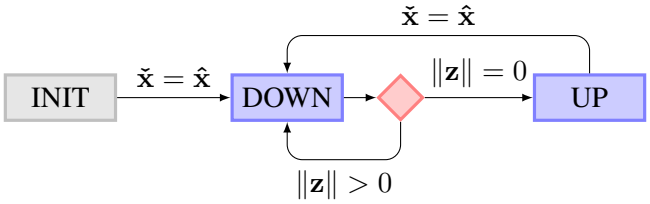
$$\text{subject to} \;\; \mathbf{A}(\hat{\mathbf{x}}+\mathbf{z})=\mathbf{1}, \;\; \hat{\mathbf{x}}+\mathbf{z}\geq 0.$$

3) Update: $\hat{\mathbf{x}} \leftarrow \hat{\mathbf{x}}+\mathbf{z}$.
4) If $\mathbf{A}[\hat{\mathbf{x}}]=\mathbf{1}$, then stop. The puzzle is solved.

D. Algorithm Summary

With the procedures defined above, a full Sudoku solver may be described by referring to the flow graph below.

Algorithm: Sudoku Solver



The arrows in the flow graph indicate control flow. The diamond shaped block is the only decision point in the algorithm (aside from checking for a solution which is done inside each block). The rectangular blocks are algorithmic procedures. The INIT procedure finds an initial feasible point $\hat{\mathbf{x}}$ which is taken as the initial starting point $\check{\mathbf{x}}=\hat{\mathbf{x}}$ for the down hill search. The DOWN procedure takes a single step in an entropy descending direction yielding a new $\check{\mathbf{x}}$. As shown in the flow graph, descent steps are repeated until the norm $\|\mathbf{z}\|$ of the step direction on the feasible set is zero. Then starting from $\hat{\mathbf{x}}$, UP takes a step in an entropy ascending direction yielding a new $\hat{\mathbf{x}}$. This is taken as the starting point for a new downward search. This process repeats until a solution is found. The stopping condition $\mathbf{A}[\mathbf{x}]=\mathbf{1}$ is checked after each update inside the procedure blocks.

V. Results

The proposed Sudoku solver based on entropy minimization was applied to 4000 puzzles across four levels of difficulty: “easy”, “medium”, “hard” and “evil”. One thousand puzzles in each category were

TABLE I

SUMMARY OF EXPERIMENTS WITH THREE SUDOKU SOLVERS

	Easy					Medium					Hard					Evil				
	F.	Time		Iter.		F.	Time		Iter.		F.	Time		Iter.		F.	Time		Iter.	
		μ	σ	μ	σ		μ	σ	μ	σ		μ	σ	μ	σ		μ	σ	μ	σ
A	0	0.19	0.04			204	0.19	0.03			750	0.19	0.03			958	0.18	0.02		
B	0	0.26	0.04			22	8.77	53.33												
C	0	0.1	0.03	0	0	0	0.13	0.06	0.22	0.49	0	0.22	0.12	0.95	1.03	2	0.29	0.25	1.45	2.02
D	0	0.1	0.01	0	0	0	0.14	0.2	0.31	1.79	0	0.3	0.54	1.75	5.13	1	0.48	1.56	3.31	14.49
E	0	0.1	0.01	0	0	1	0.25	1.01	0.53	3.92	5	1.2	3.59	2.45	8.89	0	1.5	1.73	2.89	2.97
F	0	0.1	0.01	0	0	0	0.29	0.42	0.29	0.68	1	1.01	1.21	1.46	2.04	3	1.49	1.65	2.27	2.84
G	$(\mu,\sigma) = (1.2, 1.0) \cdot 10^{-3}$					$(\mu,\sigma) = (4.7, 1.6) \cdot 10^{-6}$					$(\mu,\sigma) = (1.3, 0.1) \cdot 10^{-3}$					$(\mu,\sigma) = (1.4, 0.1) \cdot 10^{-3}$				

A	Sparse optimization
B	Binary integer linear programming
C	Entropy: DOWN(objective = $\mathbf{y}^T\mathbf{z}$, $\check{\mathbf{x}} \leftarrow \check{\mathbf{x}} + \mathbf{z}$), UP(objective = $-\mathbf{y}^T\mathbf{z}$, $\hat{\mathbf{x}} \leftarrow \hat{\mathbf{x}} + \mathbf{z}$)
D	Entropy: DOWN(objective = $\mathbf{y}^T\mathbf{z}$, $\check{\mathbf{x}} \leftarrow \check{\mathbf{x}} + 0.5\mathbf{z}$), UP(objective = $-\mathbf{y}^T\mathbf{z}$, $\hat{\mathbf{x}} \leftarrow \hat{\mathbf{x}} + 0.5\mathbf{z}$)
E	Entropy: DOWN(objective = $\|10\mathbf{y} + \mathbf{z}\|_1$, $\check{\mathbf{x}} \leftarrow \check{\mathbf{x}} + \mathbf{z}$), UP(objective = $\|10\mathbf{y} - \mathbf{z}\|_1$, $\hat{\mathbf{x}} \leftarrow \hat{\mathbf{x}} + \mathbf{z}$)
F	Entropy: DOWN(objective = $\|10\mathbf{y} + \mathbf{z}\|_2$, $\check{\mathbf{x}} \leftarrow \check{\mathbf{x}} + \mathbf{z}$), UP(objective = $\|10\mathbf{y} - \mathbf{z}\|_2$, $\hat{\mathbf{x}} \leftarrow \hat{\mathbf{x}} + \mathbf{z}$)
G	Dancing Links

generated in Matlab using Wang's puzzle generation code [19]. Three different Matlab-coded puzzle solvers were compared: the sparse optimization approach [10] which solves (6), the binary integer programming approach [9], and the entropy minimization approach of this paper. The number of puzzles that a technique could not solve are recorded in Table I, in the columns labeled "F.", along with statistics (mean and standard deviation) on running times for each of the methods. For the entropy-based method, statistics (mean and standard deviation) on the total number of iterations is also given.

A. General Remarks

From the perspective of the techniques discussed in this paper, a puzzle is particularly easy to solve if the feasible set $\mathcal{P}_9$ consists of a single point, then the puzzle is solved simply by finding a feasible point. This is easily accomplished using the INIT procedure which solves (4). In these cases, the sparse optimization (6) also solves the puzzle.

A situation in which the minimum entropy method quickly solves a puzzle is when the initial feasible point $\mathbf{x}$ falls close enough to the solution $\mathbf{x}^\star$ that the thresholding operation $[\mathbf{x}] = \mathbf{x}^\star$ solves the

puzzle. This frequently happens when the feasible set $\mathcal{P}_9$ is "small" due to a large number of given clue constraints.

Of the 4000 puzzles used, all 1000 easy puzzles were solved by the initial feasible point. In other words, the initial feasible point was either the solution or close enough so that thresholding produced a solution. In the sets of medium, hard and evil puzzles, 795, 250 and 42 were solved by the initial feasible point, respectively. These cases did not test the entropy minimization algorithm because they were solved by the initial value.

B. Remarks on Sparse Optimization

Comparing the numbers of puzzles not solved by the sparse optimization approach (6) (see line A in Table I) to the numbers of puzzles solved by the initial feasible point suggests that the sparse approach is really only finding a feasible point and that the l_1-norm minimization offers no benefit. For example, the 42 evil puzzles solved by sparse optimization were also solved by (4), the INIT procedure, without minimizing the the l_1-norm. This observation holds true in all 4000 puzzles except for one medium puzzle.

C. Remarks on Binary Integer Linear Programming

As the data in line B of Table I show, the 1000 easy puzzles were quickly solved by the binary integer programming approach. However, the time required to solve medium puzzles by this technique increased dramatically: almost 9 seconds on average for each puzzle. The Table shows that 22 out of 1000 medium puzzles were not solved. In these 22 instances, the solver exceeded its internal maximum number of iterations and exited. These 22 instances required between three and eight minutes each. The other 978 medium puzzles that were solved took less than one minute for the solution to emerge. If given enough time and iterations, the binary integer programming approach would certainly find a solution to any puzzle. The time required for this technique to solve hard and evil puzzles increased dramatically. In some cases, the solver did not return after more than an hour of processing. Therefore, we did not apply this technique to puzzles at the hard and evil levels.

D. Remarks on Dancing Links

The mean and standard deviation time (μ, σ) for solving Sudoku puzzles using the Dancing Links algorithm are shown on line G of Table I. Notice that Dancing Links solved the easy, hard and evil puzzles in about 1.3 milliseconds (ms) with standard deviations for the hard and evil puzzles of 0.1 ms.

Interestingly, the puzzles rated medium were solved much faster with an average solution time of 4.7 microseconds (μs) and standard deviation of 1.6 μs. It is not too surprising that Dancing Links is so much faster than the other techniques because Dancing Links performs no mathematical operations. It simply examines the satisfaction or not of a hard constraint as it searches a tree in depth first order, backtracking whenever a constraint is violated. These results show that for 9×9 puzzles, this exploration is very fast.

E. Remarks on Entropy Minimization

To properly interpret the iteration statistics in Table I, we note that the initialization procedure INIT was not counted as an iteration. Each call to UP and DOWN increased the iteration count by one. The easy puzzles were solved in zero iterations because they were all solved by the initial feasible point produced by INIT. The average number of iterations required to solve the medium puzzles is less than one because 795 of them were solved in zero iterations by the initial feasible point.

Because the initial feasible point solved many of the puzzles, the proposed entropy minimization strategy was really only tested on 205 medium, 750 hard and 958 evil puzzles. Here we report on results for several different variations depending on which objective was used and the update step size. In all cases, the Newton direction in (13) was used in the DOWN and UP procedures. The maximum number of DOWN and UP steps was set to 15 of each. In some cases, increasing the maximum number of steps would allow a puzzle to be solved. For lines C and D in Table I, the inner product objective was used in the UP and DOWN procedures. For lines E and F, the l_1 and l_2 norms were used with $\alpha = 10$. In line C of the Table, the update in the UP and DOWN procedures was set to the half step $\mathbf{x} \leftarrow \mathbf{x} + 0.5\mathbf{z}$. In all other cases, the full step $\mathbf{x} \leftarrow \mathbf{x} + \mathbf{z}$ was used. None of these entropy-based approaches solves all of the puzzles, but all of the puzzles are solved by one of these four approaches. Many other variations are possible by using different search directions, mixing different objectives, and varying the step sizes in the UP and DOWN procedures. In general as the difficulty of the puzzle increases, more iterations are required to solve the puzzle. Since each iteration requires solving a convex optimization problem, the complexity is quite low. As our algorithm required the evaluation of logarithms, to avoid calculation of $\log 0$, after every update in the algorithm, probabilities below 10^{-8} were set to 10^{-8}.

F. Stochastic Sudoku Puzzles

One way to state the exact cover problem is: Given a matrix whose elements are either zero or one, find a subset of the rows of the matrix so that there is a single one in each column of the selected

			4 6 7 9		4 6 7 9	4 6 7 9	4 6 7 9	
				6 7 9	6 7 9	6 7 9		
								3
					1 4 7	1 4 7	1 4 7	
1 4 6 8	1 4 6 8						1 4 6 8	1 4 6 8
	1 2 6	1 2 6	1 4 6					
1								
		2 3 4	2 3 4	2 3 4				
	2 4 6 9	2 4 6 9	2 4 6 9		2 4 6 9			

Fig. 1. A stochastic Sudoku puzzle. The filled in cells provide clues, which are given as a probability distribution on the contents of filled cells. In this case, uniform probabilities are assumed if a cell contains more than one digit. For example, one clue is that the fourth cell on the first row may be filled by one of the digits 4, 6, 7, or 9 with equal probability.

rows. Knuth's Algorithm X, and its efficient implementation known as Dancing Links, is a depth-first backtracking search algorithm for solving exact cover problems [8].

The clues in a standard Sudoku puzzle require that certain cells be filled by certain digits. This sort of puzzle may be formulated as an exact cover problem and solved efficiently by Dancing Links. Now consider a variation on Sudoku in which clues specify probability distributions on the contents of certain cells. Figure 1 shows an example of such a puzzle taken from [1]. This sort of stochastic Sudoku puzzle cannot be directly formulated as exact cover. Without exactness in the clues, Dancing Links does not apply. However, the stochastic puzzle specifications can be accommodated in the framework presented in this paper. The proposed algorithm solved this puzzle in 2 seconds after 16 iterations. A full study of stochastic puzzles is beyond the scope of this paper.

We provide this example to suggest that alternatives to Dancing Links are valuable because many problems including stochastic Sudoku, error correction decoding, and others can not be formulated as exact cover problems. Furthermore, other techniques such as Bartlett's [9] that do guarantee a solution to exact cover and that can accommodate probabilistic specifications can be very slow. The algorithm in this paper offers modest computational complexity for Sudoku, and while it does not guarantee a solution, it does solve over 99% of puzzles in all levels of difficulty, and it can solve stochastic puzzles. Applications of our approach to error correction decoding will be reported in a forthcoming publication.

VI. Conclusion and Future Work

The Sudoku puzzle problem was formulated in terms of finding a vector of 729 probabilities obeying certain constraints. The constraint set for a given puzzle is a polyhedron with one $\{0, 1\}$ extremal point having zero entropy. All other feasible points have entropy greater than zero. Therefore, entropy minimization was proposed as a means for solving Sudoku puzzles. However, entropy can have local minima on the boundary of the feasible set. Because entropy is a concave function, an iterative procedure was developed to step over local peaks in entropy to find the extremal point solution. Each step of the procedure involves choosing a search direction, an objective function, and a step size. This iterative procedure was tested on thousands of puzzles. Linear, l_1, and l_2 objectives with the Newton search direction and full or half steps were able to solve every puzzle tested.

The framework presented here accommodates other choices for initial conditions, search directions, step sizes and objectives. Our future work in this area aims to find a combination of these that solves any puzzle. Rounding procedures offer another avenue for exploration. An alternative to the ordinary rounding used in this work, geometric rounding [20], will also be explored in future work.

References

[1] T. K. Moon, J. H. Gunther, and J. J. Kupin, "Sinkhorn solves sudoku," *IEEE Trans. Information Theory*, vol. 55, pp. 1741–1746, Apr. 2009.

[2] R. A. Bailey, P. J. Cameron, and R. Connelly, "Sudoku, Gerechte designs, resolutions, affine space, spreads, reguli, and Hamming codes," *American Math. Monthly*, vol. 115, pp. 383–404, 2008.

[3] T. K. Moon and J. H. Gunther, "Multiple constraint satisfaction by belief propagation: An example using Sudoku," in *IEEE Mountain Workshop on Adaptive and Learning Systems*, pp. 122–126, 2006.

[4] T. Yato and T. Seta, "Complexity and completeness of finding another solution and its application to puzzles," *IEICE Trans. on Fundamentals of Electronics, Communications and Computer Sciences*, vol. 86, no. 5, pp. 1052–1060, 2003.

[5] G. Kendall, A. Parkes, and K. Spoerer, "A survey of NP-complete puzzles," *Internat. Computer Games Association Journal*, vol. 31, no. 1, pp. 13–34, 2008.

[6] Wikipedia, "Sudoku algorithms." Wikipedia, The Free Encyclopedia. http://en.wikipedia.org/wiki/Sudoku_algorithms (accessed Aug. 2011).

[7] H. Simonis, "Sudoku as a constraint problem," in *Fourth International Workshop on Modelling and Reformulating Constraint Satisfaction Problems*, (Sitges, Spain), 1 October 2005.

[8] D. E. Knuth, "Dancing links," in *Millennial Perspectives in Computer Science: Proceedings of the 1999 Oxford-Microsoft Symposium in Honor of Sir Tony Hoare* (J. W. J. Davies, B. Roscoe, ed.), (Palgrave), pp. 187–214, 2000.

[9] A. Bartlett, T. P. Chartier, A. N. Langville, and T. D. Rankin, "Integer programming model for the Sudoku problem," *Journal of Online Mathematics and its Applications*, vol. 8, May 2008. Article ID 1798. Available at http://www.maa.org/joma/Volume8/Bartlett/index.html (accessed Aug. 2011).

[10] P. Babu, K. Pelckmans, P. Stoica, and J. Li, "Linear systems, sparse solutions, and Sudoku," *IEEE Signal Processing Letters*, vol. 17, pp. 40–42, Jan. 2010.

[11] R. Lewis, "Metaheuristics can solve Sudoku puzzles," *Journal of Heuristics*, vol. 13, no. 4, pp. 387–401, 2007.

[12] J. Almong, "Evolutionary computing methodologies for constrained parameter, combinatorial optimization: Solving the Sudoku puzzle," in *IEEE AFRICON*, (Nairobi, Kenya), pp. 1–6, IEEE, Sept. 2009.

[13] A. Moraglio, C. D. Chio, J. Togelius, and R. Poli, "Geometric particle swarm optimization," *Journal of Artificial Evolution and Applications*, vol. 2008, pp. 1–14, 2008.

[14] T. K. Moon, J. H. Gunther, and J. J. Kupin, "Sinkhorn solves Sudoku," *IEEE Trans. on Information Theory*, vol. 55, no. 4, pp. 1741–1746, 2009.

[15] M. Grant and S. Boyd, "`cvx` Users' Guide." Available online: http://cvxr.com/cvx/usrguide.pdf, Feb. 2011.

[16] G. D. Birkhoff, "Tres observaciones sobre el algebra lineal," *Universidad Nacional de Tacuman Revista, Serie A*, vol. 5, pp. 147–151, 1946.

[17] T. M. Cover and J. A. Thomas, *Elements of Information Theory*. John Wiley & Sons, 1991.

[18] S. Boyd and L. Vandenberghe, *Convex Optimization*. Cambridge, 2004.

[19] H. Wang, "Solve and create SUDOKU puzzles for different levels." Available online: http://www.mathworks.com/matlabcentral/fileexchange/13846-solve-and-create-sudoku-puzzles-for-different-levels, Feb. 2007.

[20] D. Ge, S. He, Y. Ye, Z. Wang, and S. Zhang, "Geometric rounding: A dependent rounding scheme for allocation problems," tech. rep., Optimization Online, 2008. Available online: http://www.optimization-online.org/DB_FILE/2008/04/1949.pdf.