An Exponential Moving Average Algorithm

Evolutionary Algorithms Applied to Sudoku Puzzles

David Haynes
Systems Design
Aclara
St. Louis, MO, USA
dhfff@mst.edu

Steven Corns
Engineering Management and Systems Engineering Dept.
Missouri University of Science and Technology
Rolla, MO, USA
cornss@mst.edu

Ganesh KumarVenayagamoorthy
Holcombe Department of Electrical and Computer Engineering
Clemson University
Clemson, SC 29634, USA
gkumar@ieee.org

***Abstract*—Techniques to reduce the search space when an optimizer seeks an optimal value are studied in this paper. A new mutation technique called the "Exponential Moving Average" algorithm (EMA) is introduced. The performance of EMA algorithms is compared to two other similar Computational Intelligence (CI) algorithms (an ordinary Evolutionary Algorithm (EA) and a "Mean-Variance Optimization" (MVO)) to solve a multi-dimensional problem which has a large search space. The classic Sudoku puzzle is chosen as the problem with a large search space.**

***Index Terms*—Computational Intelligence, Evolutionary Computation, Games, Mean-Variance Optimization, Sudoku**

I. Introduction

CLASSIC Sudoku is a game played in which numbers are arranged in a 9x9 matrix. [1] An example Sudoku puzzle is shown in Table 1.

5	9	4	7	1	6	3	8	2
6	8	1	9	2	3	4	7	5
7	2	3	8	5	4	6	9	1
1	3	8	5	4	9	2	6	7
4	5	6	2	7	8	9	1	3
2	7	9	6	3	1	8	5	4
9	4	7	1	8	2	5	3	6
8	1	2	3	6	5	7	4	9
3	6	5	4	9	7	1	2	8

Table 1 - Example Solved Sudoku Puzzle

Some number of cells are blanked out and not revealed to the game player. The player must guess the missing numbers in the matrix in such a way that:

Every row must contain the numbers 1 through 9.
Rule 1

Every column must contain the numbers 1 through 9
Rule 2

When the 9x9 matrix is divided into nine 3x3 sub matrices, the numbers 1 through 9 must be contained in each sub matrix.
Rule 3

There is also a fourth rule which perhaps often goes unspoken:

Every cell that contains an initial value which is given as part of the solution may not be altered.
Rule 4

A. Statement of the problem

Sudoku is considered an NP-Complete problem. [2] The large search space created by complex puzzles makes it a useful problem against which to apply various solving techniques.

B. Contributions of this project

This paper compares three Computational Intelligence (CI) techniques for performance and examines the Sudoku puzzle in the testing process. 1.) The conventional "Evolutionary Algorithm" (EA), 2.) "Mean Variance Optimization" (MVO) which was introduced in [3], and 3.) "Exponential Moving Average" mutation algorithm (EMA) introduced in this paper.

II. Problem Description

A search algorithm which draws symbols from a set of 9 possibilities and places them into a blank 9x9 matrix creates a space of approximately $9^{9^9} \approx 10^{77}$ possible arrangements. If some of the rules are then applied to narrow the number of legitimate possibilities it creates a search space of approximately $(9!)^9 \approx 10^{50}$. It's been calculated that when the all of the rules are enforced there are 10^{21} possible legitimate (classic 9x9) Sudoku grids. [4] [5]

When presented with such an enormous search space, the tendency of a typical EA solver is to stall and not converge.

Common Sudoku puzzle solving strategies can reduce the search space for small puzzles, but lose effectiveness as puzzle complexity increases. Solvers tend to approach a solution but not converge. They spend a lot of time finding out what doesn't work better than the local minima it has settled into before ultimately running out of time.

III. Methodology Description and Implementation

All three solvers follow a similar representation. Each method uses the same fitness function, search space reduction

Dr. Venayagamoorthy wishes to acknowledge sponsorship by the US National Science Foundation CAREER grant ECCS #0348221

U.S. Government work not protected by U.S. copyright

technique, and crossover operator. Where they vary is in the mutation operation.

A. Fitness Functions

The fitness is measured by counting the number of duplicated symbols in string "c".

1) Row fitness

$$f_{rows}(c_{ij}) = \sum_{k=1}^{9} \begin{cases} 1, if\ c_{ijk} \in B \\ 0, otherwise \end{cases} \quad (1)$$

Where "c_{ij}" is the j^{th} row of elements to be examined belonging to chromosome "i". "B" is the set of elements in string c_{ij} generated as the string is scanned according to index "k".

2) Column Fitness

$$f_{cols}(c_{ij}) = \sum_{k=1}^{9} \begin{cases} 1, if\ c_{ijk} \in B \\ 0, otherwise \end{cases} \quad (2)$$

Where "c_{ij}" is the j^{th} column of elements to be examined belonging to chromosome "i". "B" is the set of elements in string c_{ij} generated as the string is scanned according to index "k".

The fitness is measured by counting the number of duplicated symbols in string "c".

3) Box Fitness

$$f_{boxes}(c_{ij}) = \sum_{k=1}^{9} \begin{cases} 1, if\ c_{ijk} \in B \\ 0, otherwise \end{cases} \quad (3)$$

Where "c_{ij}" is the j^{th} box of elements to be examined belonging to chromosome "i". "B" is the set of elements in string c_{ij} generated as the string is scanned according to index "k".

The fitness is measured by counting the number of duplicated symbols in string "c".

1	1	1	2	2	2	3	3	3
1	1	1	2	2	2	3	3	3
1	1	1	2	2	2	3	3	3
4	4	4	5	5	5	6	6	6
4	4	4	5	5	5	6	6	6
4	4	4	5	5	5	6	6	6
7	7	7	8	8	8	9	9	9
7	7	7	8	8	8	9	9	9
7	7	7	8	8	8	9	9	9

Table 2 -- Box numbers defined

4) Cell Fitness

$$f_{Cell}(c_{ijk}) = \left(f_{rows}(c_{ijk}) + f_{cols}(c_{ijk}) + f_{boxes}(c_{ijk})\right) \quad (4)$$

Where chromosome "i" contains row "j" and column "k". With the Sudoku grid being a 9x9 matrix, it results in 81 cells to be explored.

Col. / Row.	1	2	3	4	5	6	7	8	9
1	1	2	3	4	5	6	7	8	9
2	10	11	12	13	14	15	16	17	18
3	19	20	21	22	23	24	25	26	27
4	28	29	30	31	32	33	34	35	36
5	37	38	39	40	41	42	43	44	45
6	46	47	48	49	50	51	52	53	54
7	55	56	57	58	59	60	61	62	63
8	64	65	66	67	68	69	70	71	72
9	73	74	75	76	77	78	79	80	81

Table 3 – Row numbers (in red), column numbers (in blue), and cell numbers (in green) defined

5) Chromosome Encoding and Fitness

The chromosome for the CI algorithms is encoded as a 9x9 matrix as described by Table 3. Each cell in the chromosome corresponds to a cell in the puzzle.

$$f_{chromosome}(c_i) = \sum_{j=1}^{9} \sum_{k=1}^{9} f_{cell}(c_{ijk}) \quad (5)$$

Chromosome "i" has a fitness function which is simply the sum of the constituent fitness functions of every gene in the chromosome.

Each cell in the matrix is one gene in the chromosome.

A solution which perfectly satisfies all of the rules will have a fitness function of zero. In addition to the entire chromosome having a fitness of zero; the fitness of every row, column, box, and cell will also be zero.

A fitness of zero has to be considered a stopping condition for the search.

B. Restriction of Solution Search Space

1) Reduction of the number of symbols to be tried

a) Cell Candidate Set

The search for a solution can be sped along by restricting the size of the search space to be examined. Rules 1-3 state that the search space domain is limited to integers 1 through 9 for each dimension. The legitimate gene domain set can be represented by the set "D" in (6).

$$D = \{1, 2, 3, 4, 5, 6, 7, 8, 9\} \quad (6)$$

However, we know that the application of these rules also constrains the number of possible candidates for a given cell. The search for a solution need not attempt otherwise legitimate domain values which are outside the list of possible candidates. There are those who have researched the use of Candidate Sets to simplify the puzzle and speed to a solution [6]. The candidate set for cell c_{jk} is formed by beginning with the legitimate domain set "D" and removing symbols found in row "j" and column "k" as shown in (7).

$$CandidateSet(c_{jk}) = D - M_{row}(j) - M_{col}(k) - M_{box}(j,k) \quad (7)$$

Where,

D is the gene domain set described by (6).

Mrow( ) is a function which generates the set of symbols in puzzle row "j".

Mcol( ) is a function which generates the set of symbols in puzzle column "k".

Mbox( ) is a function which generates the set of symbols in the puzzle box for cell "j,k". Puzzle box boundaries are defined by Table 2.

2) Reduction of the number of cells to be searched

The reduction in the symbols to be searched has a side effect (in relatively easy puzzles) of reducing the number of cells that must be tested. When the candidate set for a given cell is reduced to a single candidate, no further mutation of that cell need occur.

C. Chromosome Selection and Crossover

Each chromosome in the population was evaluated according to all of the fitness formulas. The chromosome with the best fitness according to (5) was selected with an elitist policy. An offspring was then generated using mutation (asexual reproduction.)

D. Mutation

Once a chromosome has been selected as the parent, the offspring are generated by mutation.

1) Cell Selection

The CI algorithms used cell fitness to influence the probability of selecting the cell(s) to be mutated. The psuedocode below assumes that a given chromosome "c_i" has been selected for mutation, and now a number of cells "m" within the chromosome must be selected for mutation. The cell is located at row "j" and column "k" within the 9x9 chromosome matrix. A modified fitness function "mff" is computed to choose the cell. The mff uses a normalized value defined by (9). "rand" is a random value in the range (0,1). Once chosen the mutation process itself is performed (and directed) by the EA, MVO, or EMA techniques.

```
Copy most fit parent as new offspring "ci".
UpperBound = The count of cells in puzzle which have
  candidateQty > 1.
m = a random number in the range [1, UpperBound ] to
  mutate.
FOR m cells
    mff(cijk) = fcell(cijk) × x̆jk × rand
    Select cell with highest mff for mutation
   Mutate selected cell using EA, MVO, or EMA technique
ENDFOR
;Offspring completed.
```

Figure 1 -- Psuedocode for cell selection

2) Mutation

Once a cell has been selected, its mutation is guided by the particular technique of the CI algorithm. Each algorithm utilizes a different technique as described below.

a) Evolutionary Algorithm (EA)

A candidate is selected at random (with uniform random distribution) from the candidate set determined for the cell in (7).

(i) Baseline

The EA was also used to generate a baseline by drawing candidates at random from the domain set (6).

b) Mean Variance Optimization Algorithm (MVO)

The MVO algorithm was introduced in [3]. Each dimension is analyzed independently from the others, and a mean and variance calculated. This in turn is used to guide the mutation process. The form of MVO introduced in [3] adds additional means (described in Figure 2) to guide the "exploration" and "exploitation" phases of the solver. The MVO technique is too involved to describe here in much detail. The reader is referred to [3] for an explanation of the symbols used. Figure 2 supplies errata to the MVO calculation.

$$si = -\ln(vi) * f_s$$
$$s1 = \begin{cases} si, if\ xBest < \bar{x} \\ si * AF, otherwise \end{cases}$$
$$s2 = \begin{cases} si * AF, if\ xBest < \bar{x} \\ si, \quad otherwise \end{cases}$$
$$h0 = (1 - \bar{x_i})e^{-s2}$$
$$h1 = 1 - \bar{x_i}e^{-s1}$$
$$hx = \bar{x_i}\left(1 - e^{-x_i' s1}\right) + (1 - \bar{x_i})e^{-(1-x_i')s2}$$
$$x_i = h_x + (1 - h1 + h0)x_i' - h0$$

Figure 2 -- MVO calculation used to guide mutation

The MVO requires that data be normalized in the range (0,1). The set of domain variables {1,2,..9} is fairly easy to transform to the range (0,1).

$$x_{jk} = \frac{(S_{jk} - D_{MIN})}{(D_{MAX} - D_{MIN})} \quad (8)$$

Where x_{ij} is a normalized value in the range (0,1) for cell jk.

S_{jk} is a Sudoku puzzle value in the range [1,9].

D_{MIN} is the domain minimum of '1'.

D_{MAX} is the domain maximum of '9'.

How would one transform a Candidate Set, (which of course contains a list of discrete, discontinuous values) to the range (0,1)? This was accomplished using the index to the Candidate Set rather than the Candidate Set itself. Furthermore, the candidate list is sometimes quite small, but always greater than or equal to one.

$$\breve{x}_{jk} = \frac{(CListIndex(S_{jk},j,k) - 1)}{(CandidateQty(c_{jk}))} \quad (9)$$

Where $\breve{x}_{jk}$ is a normalized value in the range (0,1) for $cell_{jk}$.

S_{jk} is a Sudoku puzzle value from $cell_{jk}$ candidate set.
J is the row number for S_{jk}.
K is the column number for S_{jk}.
CListIndex ( ) is a function which identifies the index position of S_{jk} within the Candidate Set for row j, column k.

c) *Exponential Moving Average (EMA)*

At this point, a new algorithm, the EMA technique is introduced. The classic EA doesn't attempt to guide the mutation process. The MVO and EVA differ from the EA in that they do attempt to guide the mutation process.

(i) EMA mutation -- theory of operation.

Very complex puzzles can lead to inordinate search times and solutions that do not converge. The solver can spend considerable computational time finding inferior solutions. These computations need not be wasted. They can be used to identify choices which tend to result in poor performance. This can then be used to inform the mutation algorithm to avoid choices that tend to perform poorly.
This is accomplished by maintaining statistics on each candidate's performance as used within each cell. Candidates that tend to perform poorly tend to not be selected as often as candidates that perform well. This helps guide the solver to a solution.

(ii) An efficient technique for maintaining a historical average fitness

We are all familiar with the geometric progression:

$$\sum_{k=0}^{n} ar^k = ar^0 + ar^1 + ar^2 + \cdots + ar^n \tag{10}$$

When "n" is very large it approximates the convergent value where $r < 1 \; and \; n \to \infty$:

$$\sum_{k=0}^{n} \mathrm{ar^k} = \frac{a(1-r^{n+1})}{1-r} \approx \frac{a}{1-r}$$

Then, with

$$a = 1 - r \tag{11}$$

With (11) applied to (10), the series converges to "1".

Consider the fitness history $f_1, f_2, f_3, f_4, \ldots, f_t$ entrained in the geometric series which converges to one:

$$f_1 ar^t + f_2 ar^{t-1} + f_3 ar^{t-2} + \cdots + f_{t-1} ar^1 + f_t ar^0 \tag{12}$$

With r < 1, the contribution of the oldest history, f_1 will be quite small, and the contribution of the recent history f_t relatively significant.
Equation (12) can be rearranged as a historical accumulation encompassing historical fitness measurements ($f_1 \ldots f_{t-1}$) multiplied by the terms ar^t through ar^1, plus the most recent fitness term (f_t) multiplied by ar^0.

$$ExponentialMovingAverage$$
$$= (f_1 ar^{t-1} + f_2 ar^{t-2} + f_3 ar^{t-3} + \cdots + f_{t-1} a)r + f_t a$$
$$h_t = h_{t-1} r + f_t a$$

$$h_t = h_{t-1} r + f_t (1 - r) \tag{13}$$

The Exponential Moving Average is known to be a very efficient means of forecasting an outcome. [7] It retains useful knowledge but ages it – giving precedence to recent findings.

(iii) EMA applied to mutation

```
WHILE (solution not obtained or maximum number of
iterations not reached)
  Perform CI algorithm which uses an elitist strategy to retain
    the most fit chromosome …
  Compute cell fitness functions (1) through (5) including
    normalized cell fitness fc and
    AverageNormalizedCellFitness f̄c.
  IF (no improvement has occurred in the last iteration)
  THEN
    FOR each chromosome in population
      FOR each candidate used in each cell
        ht=ht-1*r+fc*a   ;update historical average
      ENDFOR
    ENDFOR
  ELSE  ;reset historical average so all are the same
    ht= f̄c for current evaluation
  ENDIF
ENDWHILE
```

Figure 3 -- Psuedocode for EMA historical average calculation

The cell's historical average fitness h(i) is normalized over the range (0,1). Mutation is performed by using the historical moving average and a random function to create a "modified fitness function" (mff). The mffs are then sorted to identify the candidate with the best fitness:

```
EMA_Contribution = rand
FOR every candidate of selected cell "i"
  mff(i) = ht(i) + EMA_Contribution × rand
ENDFOR
Sort "mff(i)" to obtain most fit selection.
Use most fit candidate as mutation value.
```

Figure 4 -- Psuedocode for EMA mutation

IV. Tests and Results

A. Testing the algorithms for efficacy

The EA, MVO, and EMA solvers were tested for their ability to solve Sudoku puzzles. The results are summarized in the following sections.

In order to get comparable test results, all algorithms were set to operate in the same manner:

- The population of all CI algorithms was limited to '2'. (Classic EA ordinarily has a much larger population in order to facilitate crossover.)

- All algorithms performed crossover by relying on asexual reproduction from a single parent.
- A large number of evaluations were allowed during each trial (10,000.)
- Fifty (50) trials were performed for each test and the results averaged.

1) EA performance when drawing from the domain set

In order to generate a baseline for comparison, an Evolutionary Algorithm was used to search each blank cell over the entire domain space {1,2,3…9}. Each candidate was selected uniformly at random. The results are summarized in Table 4.

# of blanks	Best fitness				Trial Size	
	Min	Mean	σ	Max	Mean	σ
10	0	0.8	1.21	3	3873	4412
20	0	10.4	9.989	35	8210	3195
30	9	24.16	6.783	36	10k	0
40	26	35.68	5.998	50	10k	0
50	23	40.58	6.32	57	10k	0
60	31	45.1	5.69	57	10k	0
80	33	44.22	5.19	56	10k	0

Table 4 -- Results of EA solver searching for "n" randomly imposed blanks in a Sudoku puzzle and drawing from the domain set {1..9}

All of the subsequent searches restrict the domain space that is searched. Rather than searching all symbols {1..9} for each cell as described by (6), a smaller set, the restricted candidate set (7) is searched instead. The mutation mechanism varies between the EA, MVO, and EMA algorithms.

2) EA performance when drawing from the Candidate Set

a) Convergence Overview

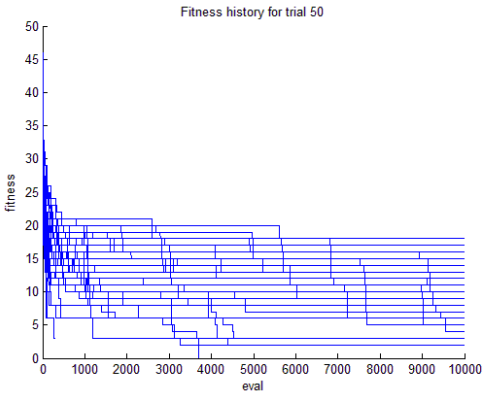

Figure 5 -- EA progress in solving a puzzle with 40 blanks over the course of 50 trials and drawing from the Candidate Set

b) Summary of Results

# of blanks	Best fitness				Trial Size	
	Min	Mean	σ	Max	Mean	σ
10	0	0	0	0	1	0
20	0	0	0	0	1	0
30	0	0	0	0	228	323
40	0	9.9	5.63	18	8.9k	2.8k
50	11	22.76	4.7	32	10k	0
60	13	31.04	5.21	43	10k	0
80	31	42.42	5.62	58	10k	0

Table 5 -- Results of EA solvers searching for "n" randomly imposed blanks in a Sudoku puzzle and drawing from the Candidate Set

3) MVO performance when drawing from the Candidate Set

a) Convergence Overview

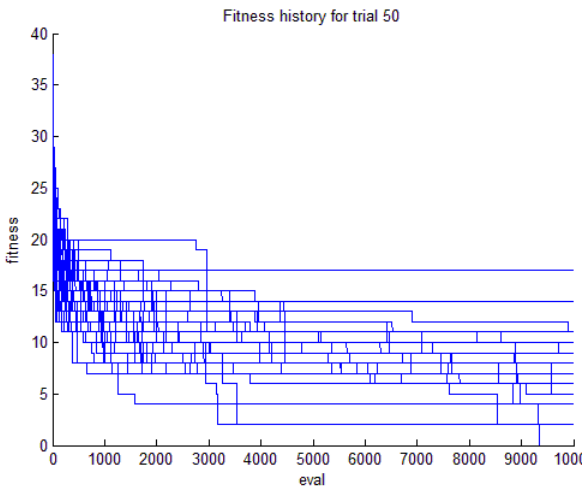

Figure 6 – MVO progress in solving a puzzle with 40 blanks over the course of 50 trials while drawing from the Candidate Set

b) Statistical Summary

# of blanks	Best fitness				Trial Size	
	Min	Mean	σ	Max	Mean	σ
10	0	0	0	0	1	0
20	0	0	0	0	64	105
30	0	0	0	0	325	420
40	0	6.4	3.9	17	9.5k	1.9k
50	9	22.9	4.39	31	10k	0
60	24	34.2	4.6	48	10k	0
80	32	44.4	5.5	58	10k	0

Table 6 -- Results of MVO solvers searching for "n" randomly imposed blanks in a Sudoku puzzle and drawing from the Candidate Set

4) EMA Performance when drawing from the Candidate Set

a) Convergence overview

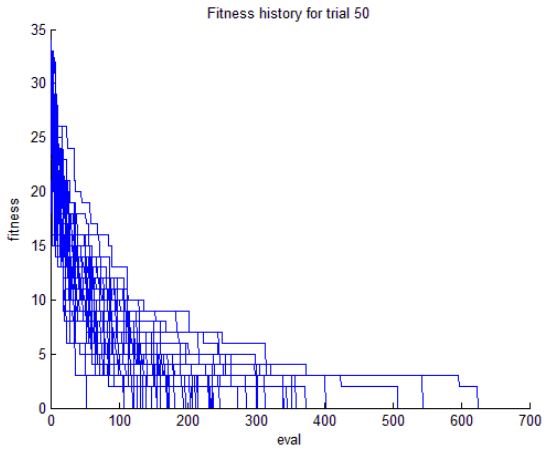

Figure 7 -- EMA progress in solving a puzzle with 40 blanks over the course of 50 trials while drawing from the Candidate Set

b) Summary of Results

# of blanks	EMA "r"	Best fitness				Trial Size	
		Min	Mean	σ	Max	Mean	σ
10	4/5	0	0	0	0	1	0
20	4/5	0	0	0	0	7.68	5.07
30	4/5	0	0	0	0	68.02	63.28
40	4/5	0	0	0	0	240.4	115.2
50	4/5	0	10.8	5.07	23	9952	337.8
60	4/5	19	26.64	3.53	32	10k	0
80	4/5	30	39.2	4.99	52	10k	0

Table 7 -- Results of EMA solvers searching for "n" randomly imposed blanks in a Sudoku puzzle and drawing from the Candidate Set

B. Comparing Solvers

Four cases were compared:

- A (baseline) Evolutionary Algorithm selecting from the Domain Set {1..9} with uniform random mutation
- An Evolutionary Algorithm selecting from the Candidate Set with uniform random mutation
- An MVO mutation algorithm selecting from the Candidate Set
- An EMA mutation algorithm selecting from the Candidate Set

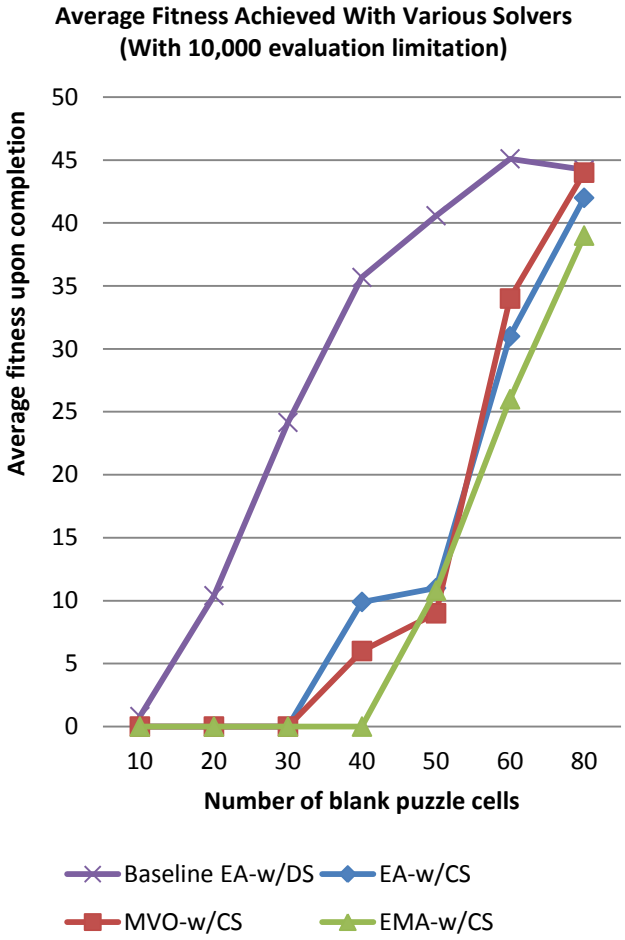

Figure 8 -- End result comparison

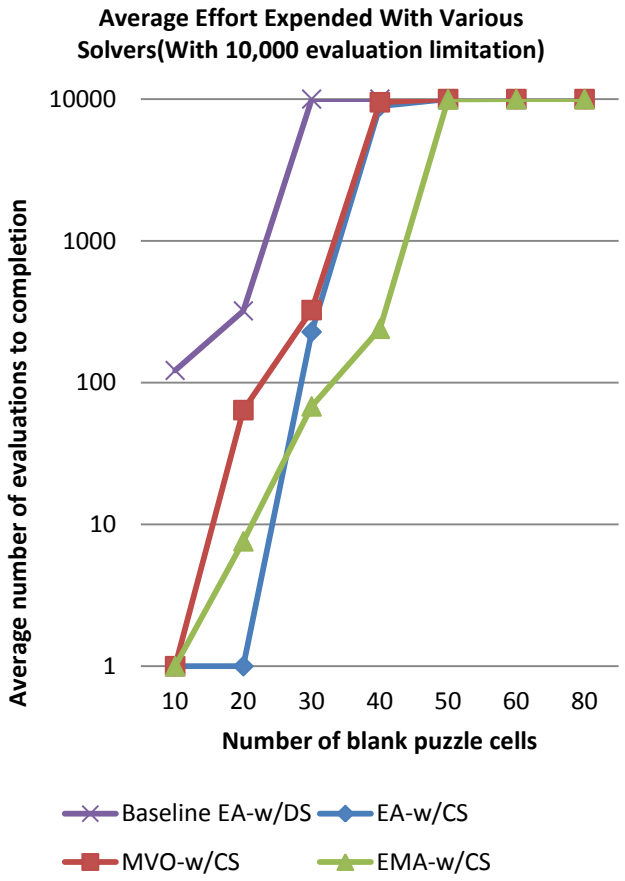

Figure 9 -- Effort comparison

V. Discussion of Results

The space to be searched by a Sudoku solver can be quite large. A search over the entire candidate domain space actually increases the search space considerably. While there are 10^{21} puzzles, there 10^{77} combinations to be tested for a puzzle with 80 blank cells. Big improvements in speed can be obtained by restricting both the number of cells which are experimented with and restricting the number of symbols which must be tried within each cell.

# of blanks	Nine symbol combinations to be explored using careless search	"Candidate Set" combinations to be explored
10 (easy)	9*9=81	1
30 (moderate)	$9^{3.3^9} = 10^{28}$	$4.4^{3.3^9} = 10^{19}$
80 (difficult)	$9^{9^9} = 10^{77}$	$(9^8 * 8)^8 * 8 = 10^{69}$

Table 8 – Approximate search space size as a function of puzzle complexity

A. Effect of restricting the search space

By the time puzzles reach a nearly undefined condition with nearly all the cells blank, the search space reaches astronomical proportions. The challenge becomes limiting the search space to the 10^{21} possible puzzles. The techniques used herein do well to limit the search space to approximately 10^{69}.

1) The "Candidate Set" technique

The "Candidate Set" technique has the beneficial side effect of limiting the symbols to be tried to a smaller set, and also the number of cells that must be experimented on. This is perhaps best demonstrated by example.

Table 9 shows 20 randomly selected cells (in red) and the resulting Candidate Set Quantity for each cell. Notice that the vast majority of the cells have a single viable candidate. This candidate is computed at the outset prior to optimization.

1	1	1	1	1	1	1	2	3
1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	2	2

Table 9 -- Example Candidate Quantity for 20 randomly selected cells

However, the savings do not hold out as the number of blanks cells increase. Figure 10 shows diminishing returns, and eventual loss of benefit as the number of blank cells reaches 60 (74%).

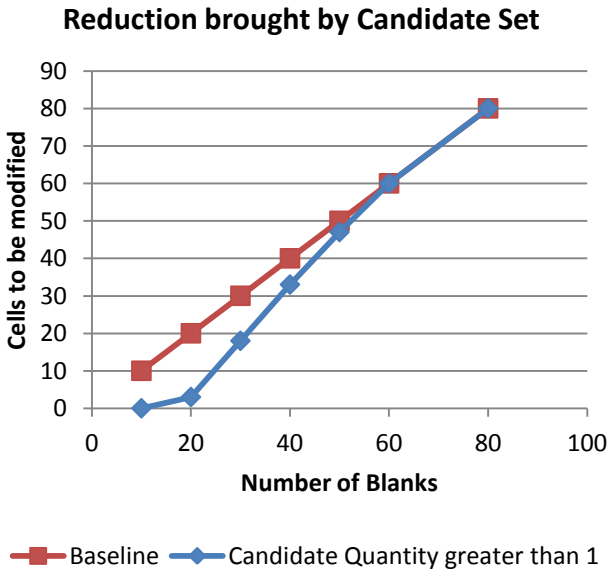

Figure 10 -- Comparative search space

These savings are then seen in Figure 8 as the difference between the "Baseline EA-w/DS" and "EA-w/CS" curves.

B. Comparing mutation techniques

All three techniques use genetically inspired CI techniques. None of them guarantee convergence. Figure 8 shows that the techniques that attempt to guide the mutation process perform better than the technique that does not. The MVO performs an analysis of the population at hand to compute the direction of the mutation. The EMA performs an analysis of the recent historical population to synthesize individual performance statistics. The EMA statistics could be viewed a Probability Distribution Function which guides the selection of the candidate. The mutation technique proposed in Figure 4 selects a random "EMA_Contribution" to apply to every mutated gene in the offspring. The use of a random contribution seemed to serve the Sudoku optimization well. For problems which are more linear and predictable in nature, a constant value could be used which is a function of the progress toward resolution.

One pitfall of the EMA approach is that rapid wholesale mutation of the chromosome in a problem in which dimensions are interrelated could produce an irrelevant history, and thus a series of choices which are no better than uniform random selection.

For this particular problem a sensitivity analysis to the value of "r" finds that it matters very little as long as $0.1 < r < 0.9$. As "r" falls within $0 < r < 0.1$ or $0.9 < r < 1.0$ it tends to lose its effect. It could be argued that because the "r" value affects the depth of history retained in the average, that "r" in (13) should increase over the course of the search as the solver transitions from "exploration" to "exploitation." This didn't appear to be necessary because the data itself stabilizes as the solution becomes apparent. The history becomes relatively consistent as the solver converges on a solution, so looking at a deeper or shallower history isn't necessarily helpful.

Another comparison is that the MVO algorithm introduced in [3] has a mechanism to convert discrete values into a normalized continuous range. Some effort was needed to make it work for discrete values. The EMA algorithm introduced here works naturally with discrete domain values. A discrete-to-continuous transformation would be needed to apply the technique to continuous domain values. One such approach was developed in [3], is shown in Figure 2, and could be used for continuous applications of the EMA.

VI. Conclusion and Future Work

Imposing constraints to reduce the search space is found to hold value in efficiently finding solutions in much the same way as learning from ones failures. The EMA algorithm introduced here attempts to identify dimension candidates which appear to perform poorly and then avoid them.

The nature of the Sudoku puzzle is such that the dimensions are interrelated. Of all of the fitness functions available, individual cell fitness was used to guide the EMA. Future work could involve study of a problem in which the dimensions are independent of each other, and the use of the overall chromosome fitness to guide the EMA. Future work could also involve the application of the EMA technique to a problem which has continuous variables. The technique should also be tested on a problem which has multiple minima. A larger chromosome population than used here should be used where multiple minima exist.

It is interesting to note that while CI techniques are genetically inspired, the forecasting and selection technique presented here of an "exponential moving average" finds its roots in the world of finance. [8] The technique was found to be useful here in guiding the mutation process away from choices which have shown to be unproductive. There are many other forecasting techniques from the world of finance that could be applied to CI mutation and candidate selection. [9] [10] [11] [12] Ultimately we find that when searching a large space it is advisable to reduce the search space where possible, and guide the mutation process to avoid unproductive search areas.

VII. Acknowledgement

The primary author would like to thank Benjamin Hammond for his promotion of the "exponential moving average" within Aclara as a means of measuring average signal strength.

VIII. References

[1] Chevron Corporation, "Suduko Daily," 2011. [Online]. Available: http://www.sudokudaily.net/instructions. [Accessed: 30-Sep-2011].

[2] L. Aaronson, "Sudoku Science," *IEEE Spectrum*, vol. 43, no. 2, pp. 16–17, Feb. 2006.

[3] I. Erlich, G. K. Venayagamoorthy, and N. Worawat, "A Mean-Variance Optimization algorithm," in *2010 IEEE Congress on Evolutionary Computation (CEC)*, 2010, pp. 1–6.

[4] B. Felgenhauer and F. Jarvis, "Enumerating possible Sudoku grids," 20-Jun-2005. [Online]. Available: http://www.afjarvis.staff.shef.ac.uk/sudoku/sudoku.pdf.

[5] Shanchen Pang, Eryan Li, Tao Song, and Peng Zhang, "Rating and Generating Sudoku Puzzles," in *2010 Second International Workshop on Education Technology and Computer Science (ETCS)*, 2010, vol. 3, pp. 457–460.

[6] J. S. Provan, "Sudoku: Strategy versus Structure," *The American Mathematical Monthly*, vol. 116, no. 8, pp. 702–707, Oct. 2009.

[7] H. Charles C., "Forecasting seasonals and trends by exponentially weighted moving averages," *International Journal of Forecasting*, vol. 20, no. 1, pp. 5–10.

[8] F. R. Johnston and P. J. Harrison, "Discount Weighted Moving Averages," *The Journal of the Operational Research Society*, vol. 35, no. 7, pp. 629–635, Jul. 1984.

[9] J. Taylor, "Smooth transition exponential smoothing," *Journal of Forecasting*, vol. 23, no. 6, p. 385, Sep. 2004.

[10] E. Uysal, F. Trainer, and J. Reiss, "Revisiting Mean-Variance Optimization," *Journal of Portfolio Management*, vol. 27, no. 4, p. 71, 2001.

[11] O. L. V. Costa and R. B. Nabholz, "Multiperiod Mean-Variance Optimization with Intertemporal Restrictions," *Jrnl of Optimization Theory and Apps*, vol. 134, no. 2, pp. 257–274, Jun. 2007.

[12] H. Markowitz, "Portfolio Selection," *The Journal of Finance*, vol. 7, no. 1, pp. 77–91, Mar. 1952.