Behind the scenes of Sudoku: Application of genetic algorithms for the optimal fun.

Thomas Bridi

Thomas.bridi@studio.unibo.it

Abstract. This work discusses about algorithms belonging to the branch of artificial intelligence for the generation of Sudoku puzzle. It will be demonstrated how the use of algorithms related to the constraint programming and genetic algorithms can improve the generation of puzzles in order to make the game more competitive and therefore more attractive to the public. In particular, it will be used an algorithm similar to the forward checking for the generation of a population of deterministic puzzles with a feasible solution and then will be used a genetic algorithm to evolve the population in order to optimize a function that rates the difficulty of their resolution.

Keywords: genetic algorithms, constraint programming, optimal solution, Sudoku generation.

1 Introduction

In this work we study the case of using artificial intelligence techniques for the generation of Sudoku, in particular genetic algorithms; we will see how these techniques can be used in areas such as journals specialized in this game to try to improve the player experience.

Chapter 2 will get a brief overview of the Sudoku game, its characteristics and the difference between probabilistic and deterministic Sudoku.

In chapter 3 we will study how the problem on generating Sudoku can be mapped as constraint satisfaction problem and we will see the limit of this kind of approach.

Chapter 4 gives a brief overview of genetic algorithms and how they are used for finding optimal solutions.

In chapter 5 we will see some related work about Sudoku and genetic algorithms.

In chapter 6 we will study the problem of generating more difficult Sudoku with the aids of genetic algorithms.

In chapter 7 we will make considerations about the results obtained from the algorithm proposed above.

2 The Game of Sudoku

As well known the Sudoku is a logical game. First of all we have to introduce some technical definitions: we will call cell the box where we are going to insert a number, grid the set of 81 cells, placed in 9x9, that makes up the Sudoku; sub-grid is a set of 3x3 cells, then we have nine rows and nine columns. The games starts with a partially filled grid and the target is to fill the remaining cells with digits from 1 to 9, constraints are that each number must appear only once in each row, column and sub-grid.

We can have two kind of Sudoku: deterministic and probabilistic. Probabilistic Sudoku is when a starting grid can be filled in different ways and more than one way reach to the objective (i.e. an empty grid is a probabilistic Sudoku). From now on we will skip the case of probabilistic Sudoku for investigating further issues related to deterministic Sudoku.

A deterministic Sudoku consists of an initial grid, partially filled, which admit a unique solution; this is a big advantage for the player that at each step will always have a valid move to do, unlike the probabilistic version that in some steps can admit more possible moves that could lead to several different solutions or to an invalid grid.

3 Generating a Sudoku using Constraints Satisfaction Problems

As we have seen previously in the definition of Sudoku we deal with constraints, then it seems logical to map the problem as a constraints satisfaction problem and it works perfectly. This is an example of the algorithm for the puzzle generation in pseudo language:

```
given G an empty grid;
do
  update the constraints of each cell;
  C := random selected cell from G;
  PossibleValues := values satisfying the constraints of C;
  V := random element from PossibleValues;
  set V as value of C;
while G does not admit solution;
```

Where “update the constraints of each cell” and “G does not admit solution” consist to apply the constraints described by the rules of the game. In order to make a more attractive puzzle, we must make it more difficult to solve; to do this we have to modify the “G does not admit solution” function taking advantage of more sophisticated techniques to detect more complex Sudoku.

In our case we do not need backtracking because this algorithm is designed to work on a large amount of puzzles, the use of backtracking is not necessary and also goes to degrade the performance in terms of memory usage, so in case of failure the algorithm will restart from an empty grid; for simplicity we will call this ad-hoc algorithm Na-

ïve Forward Checking because it use the forward checking constraint propagation but not the backtracking.

Is intuitive to understand that in this way we will not have the security of generating more complex Sudoku but only to recognize it, then the next step is to understand how it is possible to get closer to the optimal solution of our problem, thanks to artificial intelligence.

4 An introduction to Genetics Algorithms

Genetics Algorithm is a part of Evolutionary Computation. The principle under genetic algorithms is the same principle of the evolutionary theory of Darwin: in an environment, individuals that survive and reproduce, are those with better characteristics, so their genes are partially passed on to the next generation. In evolutionary computation an individual is a possible solution of a problem, a population is a set of individuals, there is a fitness function that describe how many solutions are near to the optimal solution of the problem, every individual is obtained starting from a genotype, a genotype is the representation of all the variables that can make an individual different from another.

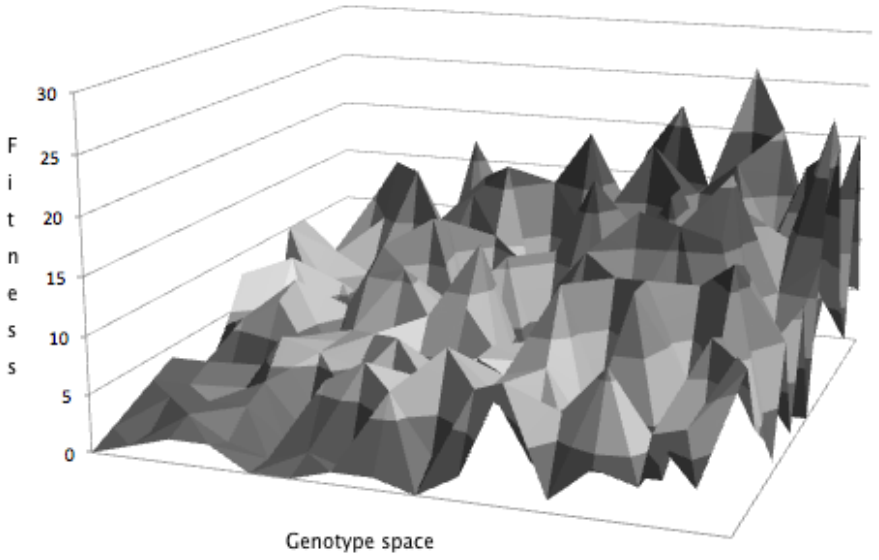

Fig. 1. Example of a fitness function of genotype space

We have a set of operations that makes a population evolve:

- **Recombination**: create a new individual from two parents.
- **Mutation**: evolve an individual modifying its genotype.

- **Insertion**: select old and new individuals to make a new population.

All these operations are done on selected individuals based on their fitness function and then theoretically every generation should get closer and closer to the optimal solution of the problem.

5 Related work

It is not easy to find words "Sudoku" and "Genetic algorithms" together in the literature especially for the generation of puzzles, M. Gold in 2005 published an application that use the approach of genetic algorithms to solve a Sudoku. In 2006 T.Mantere and J.Koljonen published a paper that describe how genetic algorithms can solve a puzzle used like a genome and also by applying swap operations; if given a blank puzzle, the algorithm fills up the grid generating a valid and solved Sudoku, but that is unplayable, the next step would be to remove the numbers from the grid to get a Sudoku with a feasible solution and the fewest number.

The approach used by T. Mantere and J. Kolijonen is completely different from the approach adopted in this work, the approach proposed for the puzzle generation use the algorithm proposed in chapter 3 for the generation of every single Sudoku but it uses a genetic algorithm for the heuristics, in particular the heuristics is about choosing to fill a mutated grid by selecting the cells in the same order used for the original grid, but using the value obtained by the mutation where possible.

6 Genetics Algorithm applied to the Sudoku generation problem

We have seen how genetic algorithms work to obtain the optimal solution. Our aim is not to get the most difficult Sudoku (undecidable problem), but to ensure that the generation of Sudoku meet a certain constraint of difficulty, common problem for anyone involved in the publication of Sudoku in magazines and games, therefore operations like recombination but especially mutation are perfectly suited to our problem.

First of all we have to decide how to set up our fitness function; as mentioned above we have several Sudoku solving techniques, we show the used techniques in the experiment in order of difficulty:

- Pencil mark
- Naked Single
- Hidden Single

Our fitness function is made to give to the puzzle a difficulty based to how many times is used a determinate techniques; every technique has a weight: Pencil mark 0, Naked Single 1 and Hidden Single 3; so a puzzle that require 2 Pencil Mark, 32 Naked Single and 7 Hidden Single moves will have difficulty 51.

A deterministic puzzle with a feasible solution is an individual of our population; its genotype is made by the list of random selected cells and their values (Chapter 3); at every step of the genetic algorithm we are going to select randomly individual on the basis of fitness and then will be applied a genotype mutation on it; finally, the mutated individuals will be reintroduced into the population.

7 Conclusions

Lastly we have made a prototype that creates first a random population of Sudoku puzzle using a Naive Forward Checking algorithm, than we have created the genotype of any individual by the filled cells and its value. The selected individuals by their fitness value has been mutated in some value in the genotype then the puzzle regenerated.

This algorithm will be repeated for a certain number of times and every times the average difficulty of the population is recalculated.

The result was that by applying the mutation, the average difficulty of the population tends asymptotically to a certain value with some small deviation from time to time; this asymptote depends on the fitness function but also by the number of techniques used by the solving algorithm.

Another test was made to calculate the percentage of successfully mutated grids and the percentage of mutation that produced a better solution, this test was done on starting population of about 40200 Sudoku grids and the result was that in average only the 8,24% of starting grid ported successfully to a deterministic puzzle before the fail of the naive forward checking algorithm; a possible way to increase this percentage could be to implement backtracking at least for the puzzle generation from genotype; the second result was that the percentage of mutation leading to a more difficult Sudoku is 35,88%.

About performance: this algorithm takes 3 hours for generating and mutating 10000 puzzle.

8 References

1. Andrea Roli: An introduction to evolutionary computation - http://lia.deis.unibo.it/Courses/AI/fundamentalsAI2011-12/lucidi/seminari/roli/intro-evolutionary_computation2012.pdf
2. John R. Koza, Riccardo Poli: A Genetic programming tutorial - http://lia.deis.unibo.it/Courses/AI/fundamentalsAI2009-10/lucidi/seminari/roli/gptutorial.pdf
3. Peter Norvig, Stuart J. Russell: Intelligenza Artificiale. Un approccio moderno.
4. E. Delucchi, G. Gaiffi, L. Pernazza: Passatempi e giochi: alla ricerca di problem e soluzioni - http://www.dm.unipi.it/~gaiffi/papers/giochi.pdf
5. Mathias Weller: Counting, Generating and Solving Sudoku - http://theinf1.informatik.uni-jena.de/publications/sudoku-weller08.pdf
6. Silvano Martello: Ricerca Operativa per la Laurea Magistrale