Solving Sudoku Puzzles with Particle Swarm Optimisation

Sean McGerty
ITEC808 student
Macquarie University
Sydney, Australia
sean.mcgerty@sudents.mq.edu.au

Abstract

This workshop paper addresses Heuristic approaches to solving Sudoku puzzles, with particular focus on Particle Swarm Optimisation(PSO). Sudoku problems and their constraints will be discussed. Heuristics used to solve Sudoku will be identified. We will then propose a component framework for supporting PSO and other Heuristics. Doing so allows us to separately detail aspects of PSO such as initialisation, optimisation, randomisation and the fitness function separately. Conclusions are drawn, implications drawn for the other Heuristics, and suggestions for further work are made.

1 Introduction

Sudoku is a popular combinatorial challenge for enthusiasts worldwide. The simple 9x9 grid and 4 constraints are easily understood, and the $6{,}700x10^{18}$ or so possible combinations ensures enough complexity to last hours.

Sudoku is also significant as a target for heuristics research. Sudoku puzzles have been shown to be NP Complete, which means you may need to check all possible combinations to know you have the best solution. Finding reliable ways to quickly solve Sudoku problems may offer improved ways to solve NP Complete problems.

Generally speaking, Heuristics solve problems by working as a population. Each member of the population randomly changes, is measured by a fitness function, and then information is shared about the more successful members.

There are two forms of Heuristics, Evolutionary Algorithms(EAs) and Swarming Algorythms(SA). EAs include Genetic Algorythms where particles simulate population lifecycles and the sharing of successful attributes from parents to children, or Simulated Annealing where elements are combined. SAs include Particle Swarm Optimisation where the particles move towards the most successful particle, in the way birds my flock in flight.

The weakness of Heuristics is a tendency to drive into local maxima. This happens because Heuristics chase the best solution, without knowing if it is really the best possible solution. A Heuristic might solve most of a puzzle, and only be able to tell it can't improve. Our aim is to improve our heuristic to avoid these local maxima fill all 81 cells more of the time.

The Heuristics have a lifecycle, and breaking their operation into components shows that there are multiple aspects to configuration and management. We suggest a component framework which will allow our PSO implementations to be optimised within each of these components.

2 Sudoku

In this section we review how Sudoku puzzles are defined.

[1]Sudoku roughly translates from the Japanese as "Solitary Number"[4]. The numbers 1-9 are distributed on a 9x9 grid using 4 contraints:

- Numbers are in the range 1 to 9.
- No number is duplicated in any row
- No number is duplicated in any column
- No Number is duplicated in a region, defined as 9 exclusive 3x3 blocks.

[1] *http://en.wikipedia.org/wiki/Sudoku*

Sudoku boards are seeded with numbers known as Givens. The lowest observed number of givens that leads to a unique solution is 17 with 47,000 of these known, but the theoretical minimum is as yet unidentified[10]. Players cannot change Givens, but vary the values in other cells in an attempt to find a solution which fills the boards and satisfies the constraints as shown in Figure 1.

Sudoku puzzles have existed in popular form since 1892[10] and are believed to be based on [2]Latin Squares[8] first presented by Leonhard Euler in 1783[4]. Euler used Latin characters rather than numbers and didn't enforce the Sudoku region constraint. Latin Squares problems have been known to be NP Complete *[11](Berlekamp, McEliece, Tilborg 1978)* for some time, while Sudoku Puzzles have also been shown to be NP Complete more recently *([1]Yato Seeta 2005)*.

[3]Given these constraints, and the size of the board, the total number of possible Sudoku combinations has been identified in rec.puzzles in 2003 as 6,670,903,752,021,072,936,960 [10]. Sudoku puzzles are often ranked for difficulty, and there appears to be no direct correlation between the difficulty of a problem and the number of givens.

3 Heuristics

In this section we review Heuristics which have been used to try and solve Sudoku puzzles.

[4]A heuristic technique is one that attempts to find good fit solutions, often by trial and error or learning techniques, and avoids the requirement to evaluate every possible combination as a brute force method would.[17]

Heuristics start with a collection of solutions which they attempt to improve. The result is the best so far, rather than the best possible solution. Our goal is to optimise heuristics so as to have confidence that the heuristic good solution is effectively the global optimum. If this were repeatable, solutions to other NPComplete problems may also be possible.

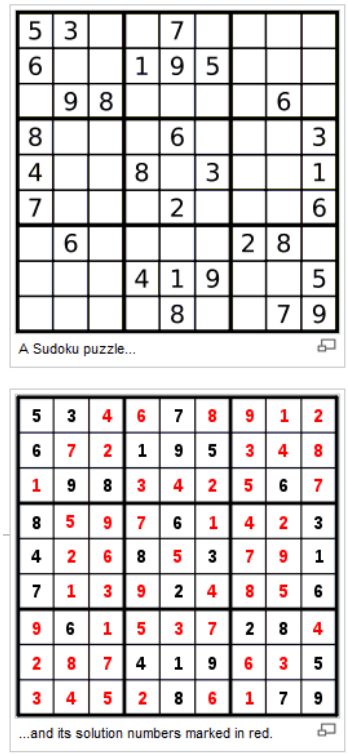

Figure 1 - Sudoku puzzle

A key optimisation for heuristics is being able to avoid local maxima. We are lucky in that fitness functions for Sudoku can identify a correct solution, and in many cases there is only one unique solution.

Heuristic solutions to Sudoku fall into 2 main categories, Evolutionary and Swarming.

3.1 Evolutionary Heuristics

[5]Evolutionary Algorythms(EAs) have generations of solutions that attempt to optimise themselves by combining the best elements of each other.

The general approach is outlined as follows:

- A generation of solutions are created
- They are assessed by a fitness function.
- The more successful members of the population share attributes or propagate.
- The least successful solutions are eliminated or attempt to gain the attributes of the more successful.

[2] http://en.wikipedia.org/wiki/Latin_square
[3] http://en.wikipedia.org/wiki/Algorithmics_of_sudoku
[4] http://en.wikipedia.org/wiki/Heuristic
[5] *http://en.wikipedia.org/wiki/Evolutionary_computation*

- A random mutation factor is applied.
- This process continues until the population stabilises around a collection of local maximums.

Genetic Algorythms are EAs based around Darwinian evolution and survival of the fittest. *[4](Perez, Marwala 2008), [5] (Mantere, Koljonen 2007), [6] Darwin 1859.*

Simulated Annealing is a type of EA which combines attributes from good solutions into weaker solutions, and is based on how mineral crystals grow. *[4](Perez, Marwala 2008)*

An important consideration for EAs is that, should they be stateless, they can be combined with other Heuristics as interleaved optimisations.

3.2 Swarming Heuristics

Swarming heuristics involves particles which attempt to improve their position by moving towards better performing neighbors.

A population of points are randomly thrown into the solution space. Each point knows its "location" and its "velocity". Points know their current solution fitness and their best solution so far. Points also know the fitness and best solution so far for their neighbors. *[12](Li, Tian, Hua, Zhong 2006)*

Neighbors can be defined by distance or relationships. Distance calculations for points can be computationally expensive. Relationships can be maintained via lookup tables. As particles converge either neighbor algorythm may yield similar results.

Each movement iteration combines the current particle 'velocity' and a randomisation factor with a range of trust factors including: current location, this particles' best solution so far, the positions of its neighbors, and the success of its neighbors.

Repulsive Particle Swarm Optimisation adds a factor for ensuring particles don't get too close together in the hope of avoiding local maxima. *[4](Perez, Marwala 2008)*

Geometric Crossovers are an important variant of PSO where the velocity component is replaced by the ability to crossover cells with other solutions. *([2]Moraglio 2007), [7](Moraglio, Di Chio, Togelius, Poli 2008).*

3.3 The Fitness Function

Evaluating the success of any solution is performed by a fitness function. The fitness function gives higher scores for more developed solutions. A successful solution therefore is the maximum allowable fitness function value.

In the case of Sudoku, if we were to count the number of placed numbers a maximum fitness function value may be 9x9x9=729 (9 degrees of freedom on each of 3 constraints). The fitness function can be reused by each heuristic tested.

3.4 Heuristic Optimisations

Optimisations for these heuristics usually try to balance speed to a solution against collecting in a local maximum.

Geometric crossovers involve swapping a subset of values between 2 solutions as permitted by the constraints.

Repulsive affects act against the tendency of solutions to gravitate to the same point.

Simulated Annealing *[3](Lewis 2007)* is sometimes used as an optimisation technique on other heuristics, and works in a similar fashion to geometric crossovers, except that values are combined rather than swapped.

Optimisations can be prioritised with factors effecting when they contribute in the solution lifecycle. For example repulsive and randomisation factors may be more significant during early stages of the solution process and less apparent the closer candidate solutions become.

4 Assessing Heuristic Components

In this section we extend from our review of sources and suggest a breakdown of the Heuristics into pieces that can be separately optimized. These components can then be reassembled in a framework improving a range of Heuristic and optimisation combinations.

Heuristic approaches appear to lend themselves to a componentisation and reuse. If the compo-

nents are inputs to the heuristics, then improving the quality of the components increases the probability of success for the heuristics.

Each heuristic: is seeded with a board defining the problem givens, is seeded with random solutions including the givens, can be defined to work with the same representation of the Sudoku board.

The fitness function has more to do with assessing the constraints than any particular heuristic.

Optimisations appear to be interleaved between heuristic iterations, and can therefore be independent.

Trust factors such as randomisation may be consistent between heuristics. As well the mechanism controlling trust factor prioritisation over the lifecycle of a heuristic would be reusable if it remained configurable per heuristic. This is described in Figure 2.

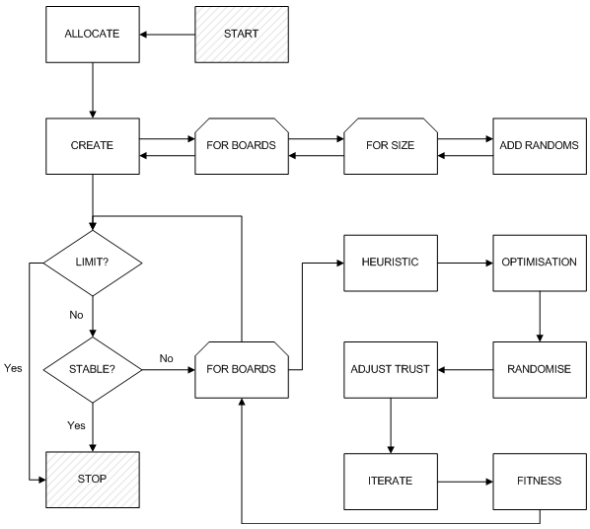

Figure 2 - Component workflow

4.1 The board and givens

The board and givens should be defined as simply as possible on a 9x9 grid. The solution encoding method appears to be a fundamental scaling factor for the number of PSO particles that can be supported and the populations under management in Evolutionary heuristics. Graph implementation of the problem space have been used, but these seem to favor individual implementations. *[8](Simonis 2005).* Indeed how to use alternative encodings is an important element in itself. *[9](Lynce, Ouaknine 2006), [10](Moon, Gunther 2006)*

Combinatorial and search based optimisations are good enough to allow brute force solutions to be achieved in sub second time. Adding these abilities to Heuristics devalues our goal, so no namespaces supporting these processes should be added.

4.2 Empty Cells and Degrees of Freedom

An optimisation that is particularly successful during brute force solutions is working with cells with the lowest degrees of freedom. Working in open areas of the board is not necessarily helpful if it can be invalidated by cells which have fewer valid options.

Identifying cells with restricted degrees of freedom can be done by search mechanisms, however this devalues the heuristic approach and we will not pursue this avenue. The Fitness Function could contain a factor that favors solutions which are using cells with more populated constraints. Filling these cells earlier on in the process increases the flexibility available to the heuristic later on in the process. This also delays the appearance of local maxima.

4.3 Random Candidate Solutions

The size of the initial random solutions is expected to be significant.

Consider the process of adding valid random numbers to a board. It is expected that the number of valid combinations remaining drops as numbers are added to the board. However the rate at which combinations are eliminated is relative to the the complexity of the problem presented by the numbers placed, as stated earlier.

Therefore it is expected that a board might transition from many possible solutions, to one valid solution, to no valid solutions, as random numbers are successively added. The point at which no more random numbers can be added matches the situation for a heuristic where a local maxima has been found. This is indicated in Figure 3.

A higher degree of randomisation in the initial population distributes the population and particles as far as possible across the namespace. It is expected that this also improves the benefits seen from optimisations such as Geometric Cros-

sovers as there are more cells to work from when comparing particles.

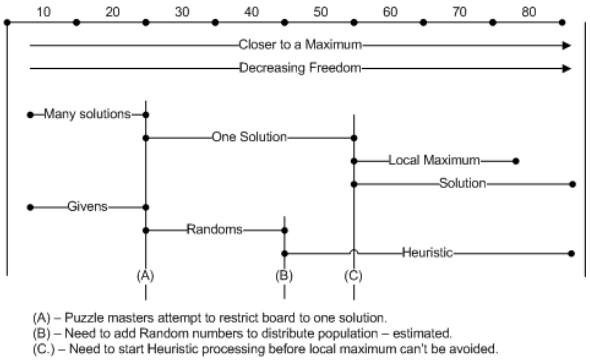

Figure 3 – Considerations for Randoms

Just adding random numbers has a very low probability of success. Continuing to add random numbers before commencing the heuristic brings the solution closer to local maxima.

It is expected the optimum size of random solutions can be validated by attempting to solve puzzles with a brute force algorithm as Randoms are added. As the transition of through the states of "many solutions" to "one solution" to "unsolvable" is expected, we can generalize a good number of target Randoms for truly random solutions.

4.4 Optimisations Interleaving Heuristics.

When we look at the namespace requirements for Evolutionary and Swarming algorithms it becomes apparent that there are similarities.

The suitability of an optimisation met for interleaving with a heuristic appears to be determined by its namespace and its behaviors:

Optimisations that do little more than work with boards are favored. It would be possible to see Simulated Annealing, or Geometric Crossovers used as Optimisations as they simply compare boards and swap values. Others like PSO which manage additional information such as velocities, prior solutions or neighborhood relations are unsuitable.

Optimisations that would need to discard solutions, such as Genetic Algorithms, may be inconsistent with the management requirements of heuristics that try and improve a solution over time like PSO. To continue to operate PSO would need to reset the particle from the deleted solution to the new one, which looses the value of the particle knowing its best solution so far.

As a result Geometric Crossovers, Simulated Annealing and Repulsive factors are candidates as optimisations because they are immediate while Genetic Annealing and PSO are not.

Evolutionary algorithms use populations of boards which vary by iteration.

Swarming Algorithms need to track: the current solution, the best solution so far, the current 'velocity', and any neighbor relationships (distance or social).

As a result it would be possible to perform an iteration of an Evolutionary Algorithm in between iterations of PSO if the EA used the current solution as its board. The reverse would not necessarily be true as the PSO's extra details wouldn't be available.

This explains why relatively stateless optimisations such as Simulated Annealing and geometric crossovers have been used as an optimisation for PSO.

4.5 Trust Factors

Each of the heuristics attempts to balance: a random factor, trust in itself now, trust in its previous best solution, trust in another population member / particle, and trust in interleaved optimisation.

These trust factors can change over time. For example it makes sense to increase randomisation earlier in the process to help distribute the particles / population. Later on in the process, as we begin to approach candidate solutions, it makes more sense to increase trust in more successful elements in the population.

This situation was evident in the Repulsive PSO analysis, where a repulsive force was added to a Swarm in an attempt to avoid local maxima. In this case the particles merely stabilised at the minimum distance from each other and the desired benefits were not realized. The stronger strategy of randomisation earlier with a more focused result later seems to have better results.

The mechanism for managing these trust factors over the lifecycle of the solution can be a component. This would allow comparisons on the effectiveness of trust factors for Optimisations and randomisation.

5 Conclusions

Particle Swarm Optimisation appears to have shown the best success rates when combined with Geometric Crossovers. Unlike Evolutionary Algorythms, the Geometric Crossovers aren't driving towards improving the solution and approaching a local maximum. The Crossovers are redistributing the particles without violating the constraints. As always, a balance is formed between the trust factors for approaching solution and the trust factors for randomisation. However in this case, randomisation extends to redistribution of the solution as well as incremental change.

The suggested component framework offers significant advantages often not apparent in reference implementations. Instantiation, lifecycle management, trust factors and interleaved optimisations can all be controlled and measured outside of any given heuristic.

5.1 Further work

A reference implementation of the component framework can be created for PSO with Geometric Crossovers. Care should be taken for development considerate of later extension. Instantiation and Trust factors can then be varied to maximise success.

Once this is completed new interleave optimisations can be developed such as Simulated Annealing. The framework configuration can again be optimised.

Finally new heuristics can be implemented within the framework, and optimisations and configurations tested / optimised for each.

6 Acknowledgements

This study is the result of a proposal by Mehmet Orgun. Content is primarily based on a sequence of papers by Alberto Moraglio on Particle Swarm Optimisation. Genetic Algorythms discussion largely comes from by Sudoku Genetic Algorythm approaches by Mantere and Koljonen. Simulated Annealing by Lewis. Generalised Stoachistic approaches by Perez and Marwala.

[1] T. Yato and T. Seta. *Complexity and completeness of finding another solution and its application to puzzles*. Preprint, University of Tokyo, 2005.

[2] Alberto Moraglio, Julian Togelius *Geometric Particle Swarm Optimization for the Sudoku Puzzle*. University of Essex, UK 2007.

[3] R. Lewis. (2007). *Metaheuristics can Solve Sudoku Puzzles*. Journal of Heuristics Archive, 13(4), 387-401.

[4] Meir Perez and Tshilidzi Marwala - *Stochiastic Optimization Approaches for Solving Sudoku* 2008.

[5] Mantere, T.; Koljonen, J. *Solving, rating and generating Sudoku puzzles with GA Evolutionary Computation, 2007*. CEC 2007. IEEE Congress on Volume , Issue , 25-28 Sept. 2007 Page(s):1382 - 1389

[6] Darwin, C.: *The Origin of Species: By Means of Natural Selection or The Preservation of Favoured Races in the Struggle for Life*, Oxford University Press, London, 1859, A reprint of the 6th edition (1968)

[7] Alberto Moraglio, Cecilia Di Chio, Julian Togelius, and Riccardo Poli - *Geometric Particle Swarm Optimization* (2008)

[8] Helmut Simonis - *Sudoku as a Constraint Problem* – 2005 Imperial College London

[9] Ines Lynce, Joel Ouaknine - *Sudoku as a SAT Problem* – 2006

[10] Todd K. Moon and Jacob H. Gunther - *Multiple Constraint Satisfaction by Belief Propagation: An Example Using Sudoku* - July 2006

[11] Elwyn R. Berlekamp, fellow IEEE, Robert J. McEliece, Henk C. A. Van Tilborg - *On the Inherent Intractability of Certain Coding Problems* - IEEE TRANSACTIONS ON INFORMATION THEORY, VOL. IT-24, NO. 3, MAY 1978

[12] Xiangyong Li, Peng Tian, Jing Hua, and Ning Zhong - *A Hybrid Discrete Particle Swarm Optimization for the Traveling Salesman Problem* – 2006