<u>Almedius: Genetic Software for Solving Sudoku</u>

CSCK502: Reasoning and Intelligent Systems

Ben Schlup, James Grant and Muhammad Ali

This report details the development of a software programme known as Almedius which solves a game similar to Sudoku with the aid of genetic algorithms. The objective is to fill a grid of size 4x4 with four different letters so that each column, each row, and each of the four sub-grids of size 2x2 contain each of the letters only once. The code will be analysed and explained and overall performance discussed in detail. Furthermore, shortcomings of the software along with possible improvements will also be discussed to provide a comprehensive critique.

1. Introduction

(Negnevitsky, 2011) defines genetic algorithms (GAs) as a class of stochastic search algorithms based on biological evolution. Figure 1 shows the structure of a genetic algorithm as it executes a sequence of events from start to stop stage.

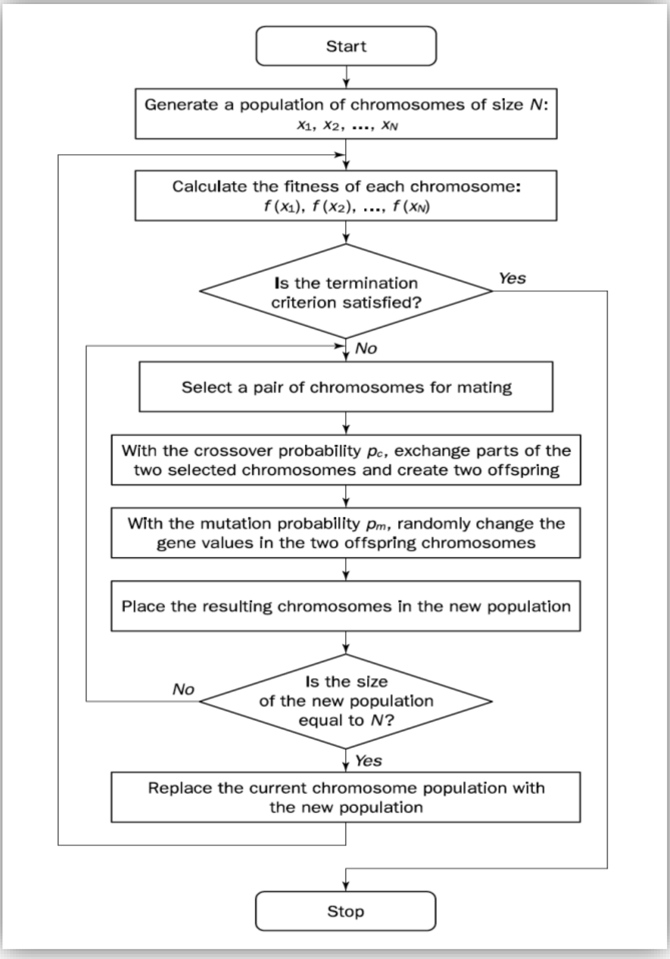
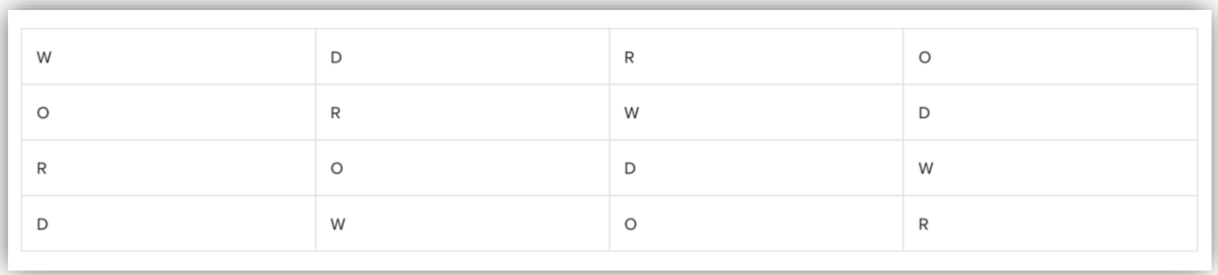

Start
Generate a population of chromosomes of size N: $x_1, x_2, \ldots, x_N$
Calculate the fitness of each chromosome: $f(x_1), f(x_2), \ldots, f(x_N)$
Is the termination criterion satisfied?
Yes
No
Select a pair of chromosomes for mating
With the crossover probability p_c, exchange parts of the two selected chromosomes and create two offspring
With the mutation probability p_m, randomly change the gene values in the two offspring chromosomes
Place the resulting chromosomes in the new population
Is the size of the new population equal to N?
No
Yes
Replace the current chromosome population with the new population
Stop

Figure 1: Schema of a generic genetic algorithm (Negnevitsky, 2011).

According to (Negnevitsky, 2011), GAs employ ***natural selection*** and genetic inspired techniques such as ***crossover*** and ***mutation*** to move from one population of artificial chromosomes to a new population; each chromosome is made up of genes and each gene is represented by 0 or 1. Furthermore, as seen in figure 1, GA represents an iterative process and each iteration is known as a ***generation***. (Negnevitsky, 2011) also states that a typical generation can vary from 50 to over 500 and an entire set of generations is known as a ***run***.

As GAs are modelled after nature, (Negnevitsky, 2011) argues that the fundamental principle of fittest species being able to survive, breed and pass their genes on to the next generation is simulated in GAs with the introduction of chromosome selection techniques which will be discussed in detail in software configuration section. (Negnevitsky, 2011) highlights the function of crossover operator as follows: a random crossover point between two parent chromosomes 'break' is identified by crossover operator and chromosomes parts after this point are exchanged to create new offspring. Similarly, the role of a mutation operator, according to (Negnevitsky, 2011), is to exchange or flip randomly selected gene in a chromosome with the goal of improving average fitness of the population and prevent the search algorithm from getting trapped on a local optimum. Summarising the roles of genetic inspired techniques, (Bala, 2017) states that crossover and mutation operators are tasked with exploration of the search space whilst selection techniques are responsible for reduction of the search area within population by disregarding poor solutions.



W	D	R	O
O	R	W	D
R	O	D	W
D	W	O	R

Figure 2: An example of solved solution according to assignment requirements.

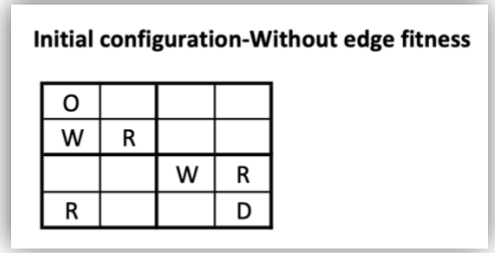
Figure 2 shows a solved grid with four different letters (W,O,R,D) satisfying the letter placement restrictions discussed earlier in the introduction. The next section will discuss the system configurations in detail.

2. Software configuration

In this section the following system parameters will be discussed in detail: fitness function, input/output encoding, crossover and mutation operators and selection strategy. Snippets of pseudocode created by the authors will aid in explaining concepts and implementations.

For a 9x9 Sudoku, Sato and Inoue (2010) suggest a block-preserving algorithm to avoid crossover operations which destroy valuable fitness in sub-blocks. Their approach has been adopted by Almedius after an implementation with a linear chromosome design and two crossover points proved to be inefficient.

Encoding of input (i.e. initial configurations) and output is using a two dimensional array of characters, illustrated in figure 3. For production use, bitstring encoding and bitwise operators should be adopted to ensure computational efficiency.


Initial configuration-Without edge fitness

O			
W	R		
		W	R
R			D

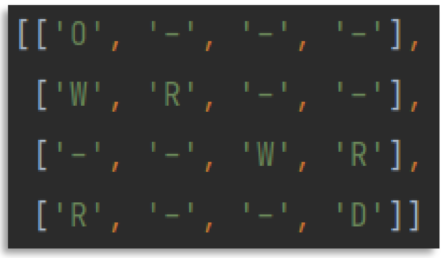

```
[['O', '-', '-', '-'],
 ['W', 'R', '-', '-'],
 ['-', '-', 'W', 'R'],
 ['R', '-', '-', 'D']]
```

Figure 3: Almedius's initial configuration and encoding in row format as Python array

The Almedius Chromosome class internally employs a chromosome design with Sudoku sub-blocks as secondary dimension of an array. Class methods allow access to the chromosome with row and column views as required by other methods (figure 4).

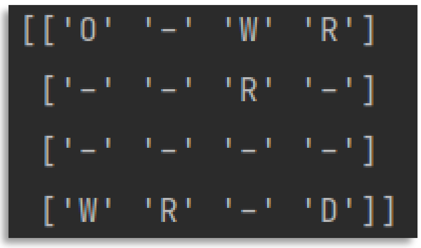
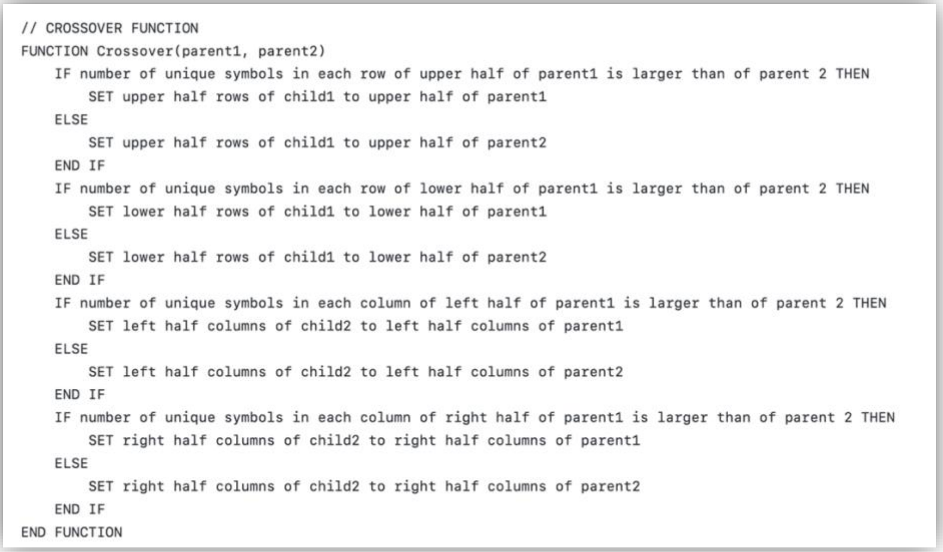

```
[['O' '-' 'W' 'R']
 ['-' '-' 'R' '-']
 ['-' '-' '-' '-']
 ['W' 'R' '-' 'D']]
```

Figure 4: Almedius internal, block-oriented encoding of chromosomes (initial configuration example from Figure 3).

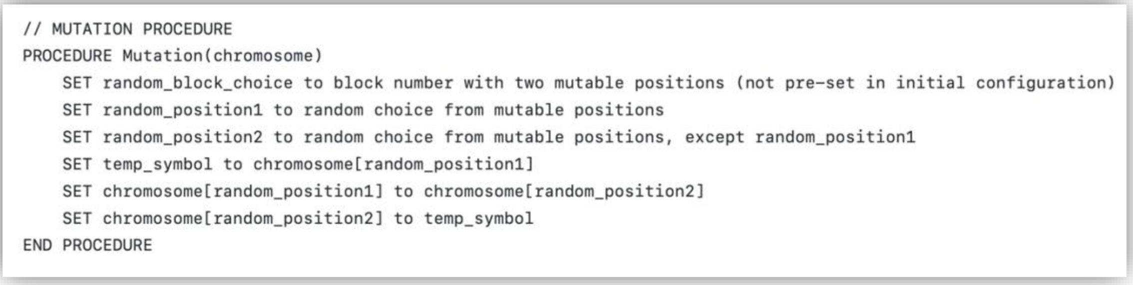
Similar to Sato and Inoue (2010), crossover operations create two offspring from every set of two parents. Offspring 1 inherits the best upper and lower halves of the two parents, and offspring 2 inherits the left and right part of two parents (see Figure 5) .



```
// CROSSOVER FUNCTION
FUNCTION Crossover(parent1, parent2)
    IF number of unique symbols in each row of upper half of parent1 is larger than of parent 2 THEN
        SET upper half rows of child1 to upper half of parent1
    ELSE
        SET upper half rows of child1 to upper half of parent2
    END IF
    IF number of unique symbols in each row of lower half of parent1 is larger than of parent 2 THEN
        SET lower half rows of child1 to lower half of parent1
    ELSE
        SET lower half rows of child1 to lower half of parent2
    END IF
    IF number of unique symbols in each column of left half of parent1 is larger than of parent 2 THEN
        SET left half columns of child2 to left half columns of parent1
    ELSE
        SET left half columns of child2 to left half columns of parent2
    END IF
    IF number of unique symbols in each column of right half of parent1 is larger than of parent 2 THEN
        SET right half columns of child2 to right half columns of parent1
    ELSE
        SET right half columns of child2 to right half columns of parent2
    END IF
END FUNCTION
```

Figure 5: Almedius crossover function (pseudo code).

The mutation procedure swaps two positions within a randomly chosen sub-block which were not pre-set in the initial configuration (Figure 6).



```
// MUTATION PROCEDURE
PROCEDURE Mutation(chromosome)
    SET random_block_choice to block number with two mutable positions (not pre-set in initial configuration)
    SET random_position1 to random choice from mutable positions
    SET random_position2 to random choice from mutable positions, except random_position1
    SET temp_symbol to chromosome[random_position1]
    SET chromosome[random_position1] to chromosome[random_position2]
    SET chromosome[random_position2] to temp_symbol
END PROCEDURE
```

Figure 6: Mutation procedure of Almedius (pseudo code).

Figure 7 shows a run with a population of four chromosomes, of which only two are chosen as parents. In generation 0, open positions are randomly set, considering the symbols not yet used in a particular sub-block. The crossover procedure keeps blocks from parents in the same place, so that pre-set positions remain unchanged. A 100% mutation rate is applied to offspring, with swapped symbols depicted in red.

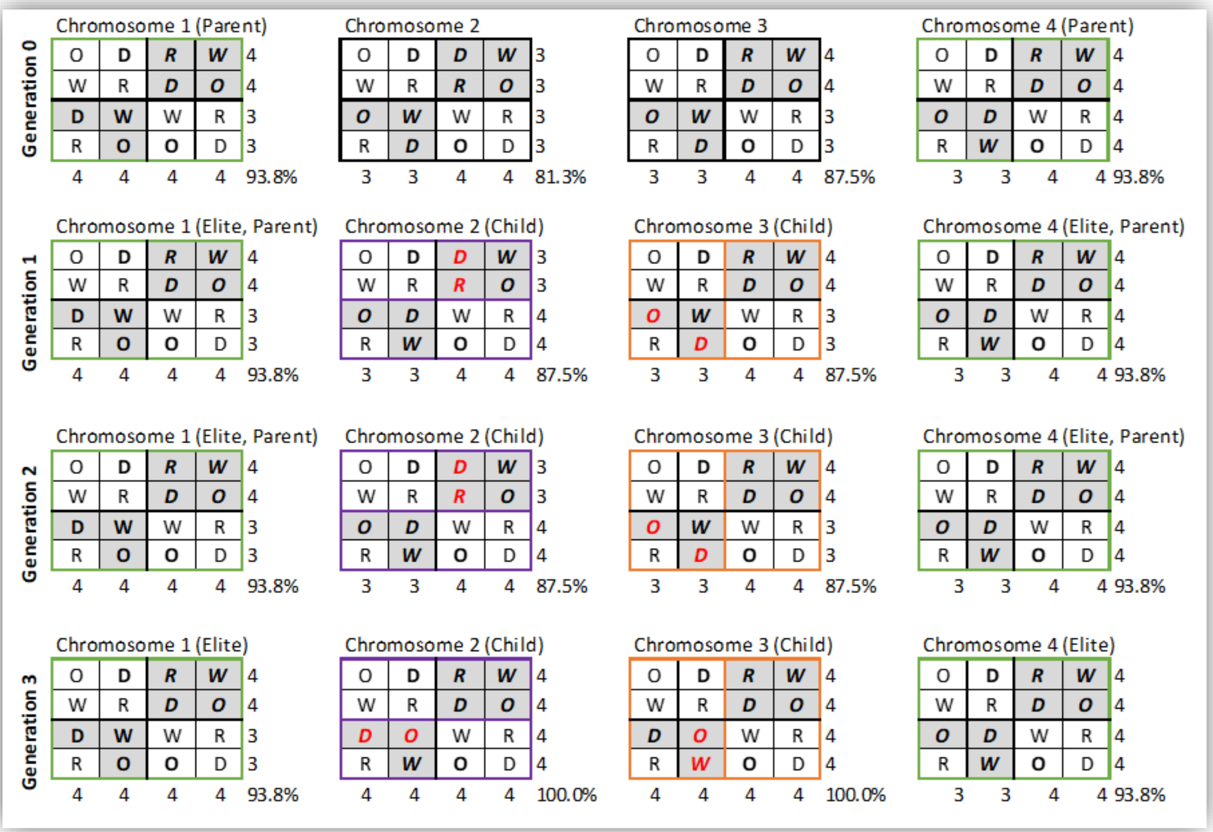


Figure 7: Example run through the algorithm.

For the case of requiring W-O-R-D spelt along one or more of the edges, the algorithm derives additional starting configurations compliant with the initial configuration. Thereafter, multiple competing populations are created (see Figure 8 for a case with two valid configurations derived from an initial configuration with two pre-sets). Extending the initial configuration has proven superior to a design where "edge fitness" was determined by the fitness function.

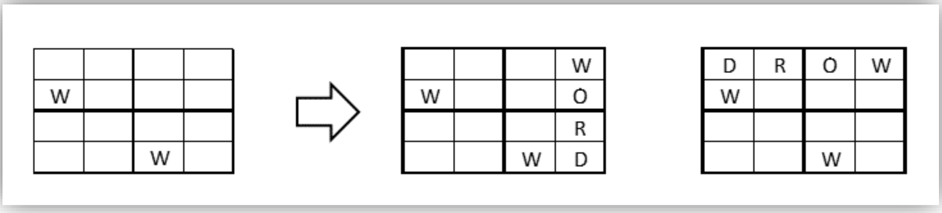
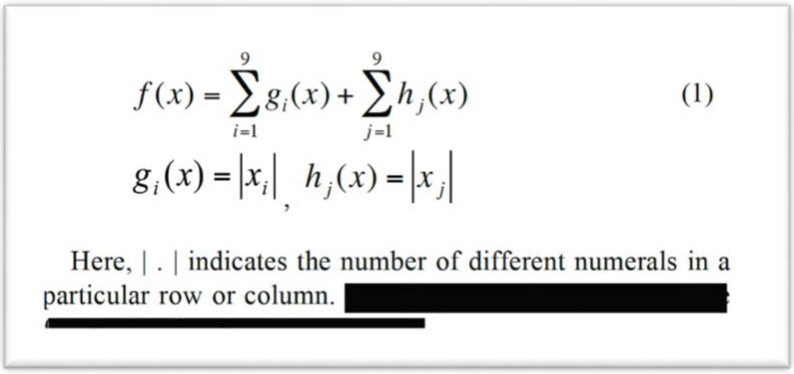


Figure 8: Derived starting configurations with "edge fitness".

According to (Kramer, 2017), fitness function measures the genetic algorithm's quality of solutions and plays an instrumental role in optimisation. After thorough literature review, the fitness function shown in figure 9 was implemented as according to (Sato & Inoue, 2010), it's been extensively implemented in solving sudoku puzzles and ensures there is no more than one numeral in a given row or column.



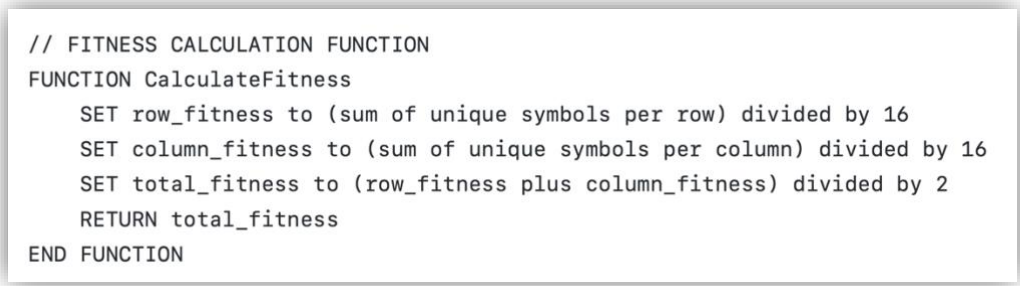
$$f(x) = \sum_{i=1}^{9} g_i(x) + \sum_{j=1}^{9} h_j(x) \qquad (1)$$

$$g_i(x) = |x_i|, \quad h_j(x) = |x_j|$$

Here, | . | indicates the number of different numerals in a particular row or column. [redacted]

Figure 9: Almedius's fitness function (Sato & Inoue, 2010).

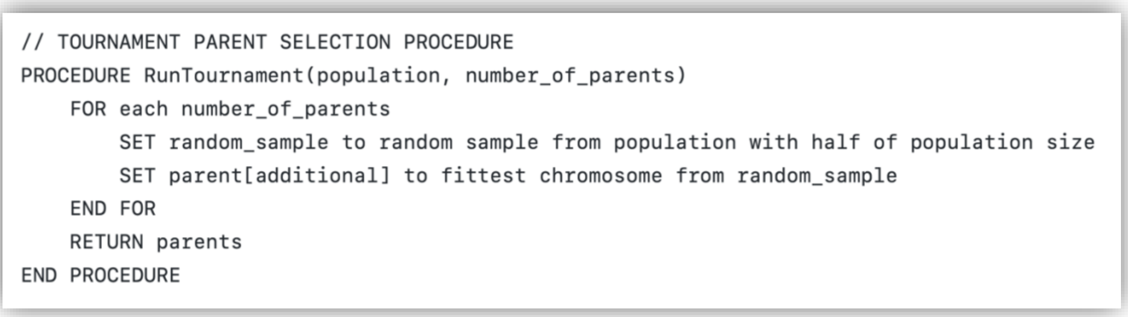
Figure 10 shows the fitness function calculation steps; row and column fitness are evaluated respectively to obtain final/total fitness for the GA.



```
// FITNESS CALCULATION FUNCTION
FUNCTION CalculateFitness
    SET row_fitness to (sum of unique symbols per row) divided by 16
    SET column_fitness to (sum of unique symbols per column) divided by 16
    SET total_fitness to (row_fitness plus column_fitness) divided by 2
    RETURN total_fitness
END FUNCTION
```

Figure 10: Fitness function calculation steps in pseudo code.

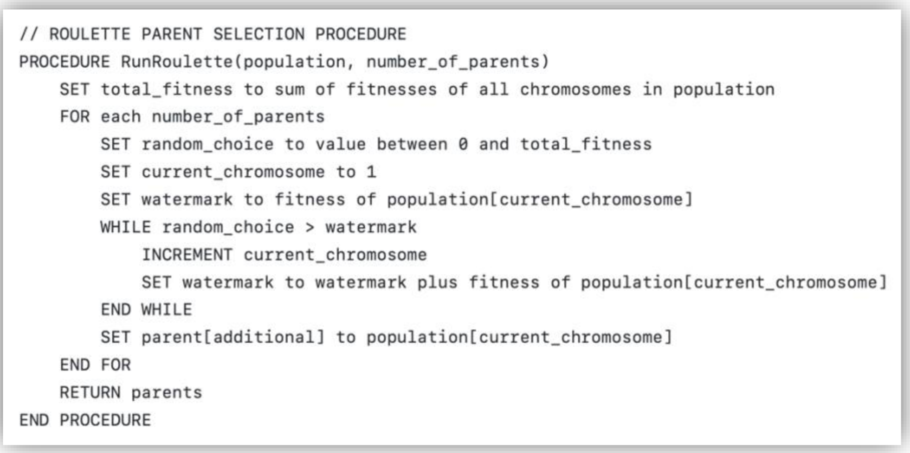
(Bala, 2017) provides a comparative analysis of selection strategies for GAs and argues that performance of GAs are evaluated in terms of convergence rate and number of generations to reach the optimal solution. Both (Bala, 2017) and (Shukla, et al., 2015) state that *Tournament selection (figure 11)* is the most popular selection technique in GAs due to its efficiency and ease of implementation.



```
// TOURNAMENT PARENT SELECTION PROCEDURE
PROCEDURE RunTournament(population, number_of_parents)
    FOR each number_of_parents
        SET random_sample to random sample from population with half of population size
        SET parent[additional] to fittest chromosome from random_sample
    END FOR
    RETURN parents
END PROCEDURE
```

Figure 11: Tournament selection technique implementation in pseudo code.

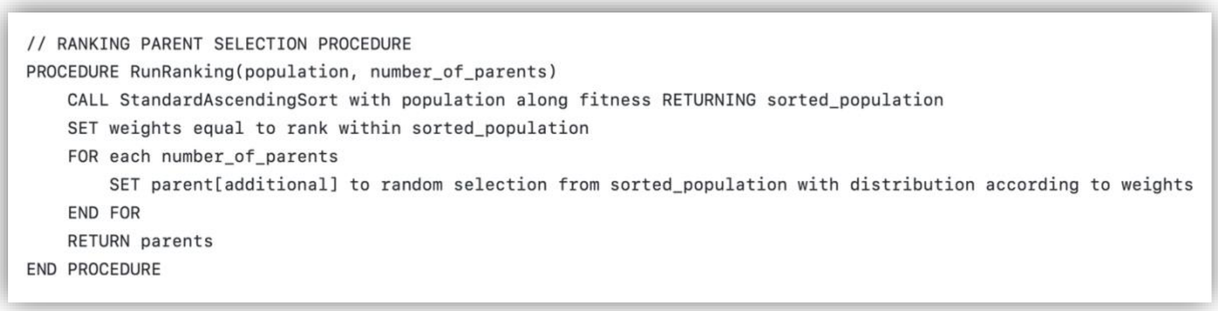
(Bala, 2017) defines *Roulette selection (figure 12)* as a technique where chromosomes are selected with a probability which is directly proportional to their fitness values. Furthermore, (Bala, 2017) also argues that as all chromosomes in. a population are given a chance to be selected, Roulette technique preserves diversity.



```
// ROULETTE PARENT SELECTION PROCEDURE
PROCEDURE RunRoulette(population, number_of_parents)
    SET total_fitness to sum of fitnesses of all chromosomes in population
    FOR each number_of_parents
        SET random_choice to value between 0 and total_fitness
        SET current_chromosome to 1
        SET watermark to fitness of population[current_chromosome]
        WHILE random_choice > watermark
            INCREMENT current_chromosome
            SET watermark to watermark plus fitness of population[current_chromosome]
        END WHILE
        SET parent[additional] to population[current_chromosome]
    END FOR
    RETURN parents
END PROCEDURE
```

Figure 12: Roulette selection technique implementation in pseudo code.

(Bala, 2017) defines *Ranking selection (figure 13)* as a technique where a given chromosome’s probability of being selected is tied to its relative fitness to the entire population. According to (Bala, 2017) and (Kramer, 2017), major advantages of this technique are avoidance of premature convergence and elimination of need to scale fitness values, although (Bala, 2017) argues that due to the necessity of sorting populations, this technique can be computationally expensive.



```
// RANKING PARENT SELECTION PROCEDURE
PROCEDURE RunRanking(population, number_of_parents)
    CALL StandardAscendingSort with population along fitness RETURNING sorted_population
    SET weights equal to rank within sorted_population
    FOR each number_of_parents
        SET parent[additional] to random selection from sorted_population with distribution according to weights
    END FOR
    RETURN parents
END PROCEDURE
```

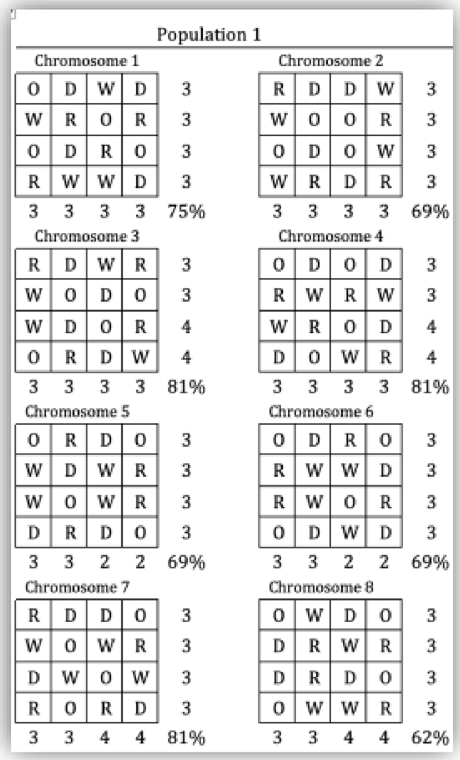
Figure 13: Ranking selection technique implementation in pseudo code.

3. Initial Tests & Results

This section will detail the tests, results and comparative analysis of selection techniques to provide a holistic performance view of Almedius.

Initial Tests

Data were gathered from the execution of a set number of games. Table 1 shows Population 1 of the RA-0-2-4-0.5 dataset. Each chromosome displays its row and column fitness, and its total fitness percentage.



Population 1

Chromosome 1						Chromosome 2				
O	D	W	D	3		R	D	D	W	3
W	R	O	R	3		W	O	O	R	3
O	D	R	O	3		O	D	O	W	3
R	W	W	D	3		W	R	D	R	3
3	3	3	3	75%		3	3	3	3	69%
Chromosome 3						Chromosome 4				
R	D	W	R	3		O	D	O	D	3
W	O	D	O	3		R	W	R	W	3
W	D	O	R	4		W	R	O	D	4
O	R	D	W	4		D	O	W	R	4
3	3	3	3	81%		3	3	3	3	81%
Chromosome 5						Chromosome 6				
O	R	D	O	3		O	D	R	O	3
W	D	W	R	3		R	W	W	D	3
W	O	W	R	3		R	W	O	R	3
D	R	D	O	3		O	D	W	D	3
3	3	2	2	69%		3	3	2	2	69%
Chromosome 7						Chromosome 8				
R	D	D	O	3		O	W	D	O	3
W	O	W	R	3		D	R	W	R	3
D	W	O	W	3		D	R	D	O	3
R	O	R	D	3		O	W	W	R	3
3	3	4	4	81%		3	3	4	4	62%

Table 1. A population of chromosomes.

The test was run from a Python script that looped through the configured parameter sets and executed the sets parallel. Table 2 and 3 list the parameters that affect the volume and variation of the data and those that don't, respectively.

Parameter	Options
Parents	2,4,8
Initial Configuration	0,4,7
Child Mutation Probability	0.5,1.0
Target Population	8,4
Selection Strategy	TO, RO, RA
Number of Games	1000

Table 2. Parameter Options that affect data.

Parameter	Options
Output Method	M
Verbosity	2

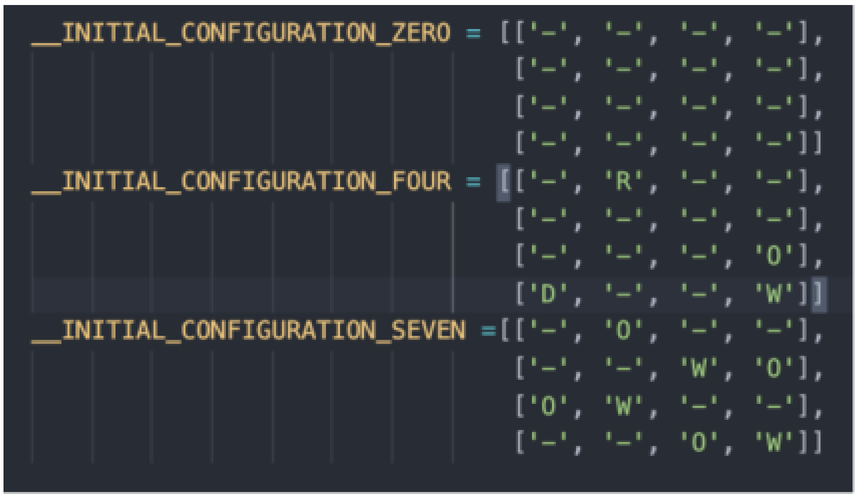
Table 3. Parameter Options that don't affect data.

Table 4 shows a subset of the parameter permutations for the batch execution, namely those of the Parents, Initial Configuration, and Child Mutation Probability parameters for the TO Selection Strategy for a Target Population of eight.

Parents	Initial Configuration	Child Mutation Probability	Target Population	Selection Strategy	Output Method	Verbosity	Number of games
4	0	0.5	8	TO	M	2	1000
8	0	0.5	8	TO	M	2	1000
4	4	0.5	8	TO	M	2	1000
8	4	0.5	8	TO	M	2	1000
4	7	0.5	8	TO	M	2	1000
8	7	0.5	8	TO	M	2	1000
4	0	1.0	8	TO	M	2	1000
8	0	1.0	8	TO	M	2	1000
4	4	1.0	8	TO	M	2	1000
8	4	1.0	8	TO	M	2	1000
4	7	1.0	8	TO	M	2	1000
8	7	1.0	8	TO	M	2	1000

Table 4. Subset of configuration values for the batch execution.

The parameters also specified one of three pre-defined initial puzzle configurations, one with no filled blocks, one with four filled blocks and one with seven. Image 1 shows the configuration of these initial puzzles in code.



```
__INITIAL_CONFIGURATION_ZERO = [['-', '-', '-', '-'],
                                 ['-', '-', '-', '-'],
                                 ['-', '-', '-', '-'],
                                 ['-', '-', '-', '-']]
__INITIAL_CONFIGURATION_FOUR = [['-', 'R', '-', '-'],
                                 ['-', '-', '-', '-'],
                                 ['-', '-', '-', 'O'],
                                 ['D', '-', '-', 'W']]
__INITIAL_CONFIGURATION_SEVEN =[['-', 'O', '-', '-'],
                                 ['-', '-', 'W', 'O'],
                                 ['O', 'W', '-', '-'],
                                 ['-', '-', 'O', 'W']]
```

Image 1. Initial puzzle configurations.

Discussing the results

- ***Generations to Converge***

The minimum Generations to Converge (GtC) obtained in the test are 0, meaning that several games achieved convergence in the first generation. The maximum GtC of 10000 represents the games that failed to converge. The mean, due to a significant number of failed generations, is higher than the median by a factor of close to 30. See table 5 for the statistical summary and figure 14 for the frequency distribution

Min	Max	Mean	Median
0	10000	291,646875	10

Table 5. Statistical summary.

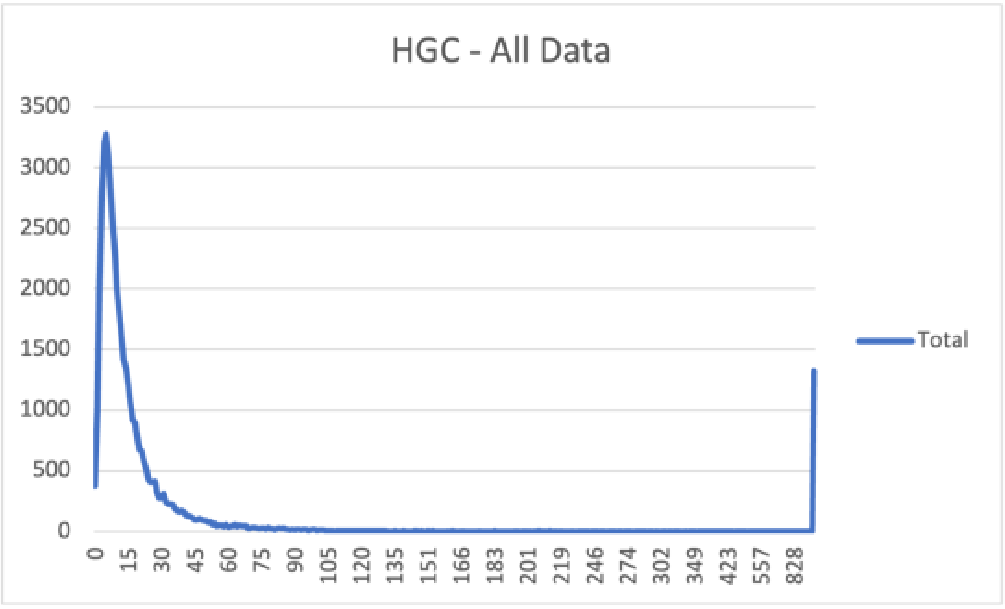


Figure 14. HGC frequency distribution.

Figure two shows a distribution that centres around 10 and has a long tail towards a sharp spike representing the cluster of failed generations. The data reveal some common characteristics of the sets that included failed generations, see figure 15.

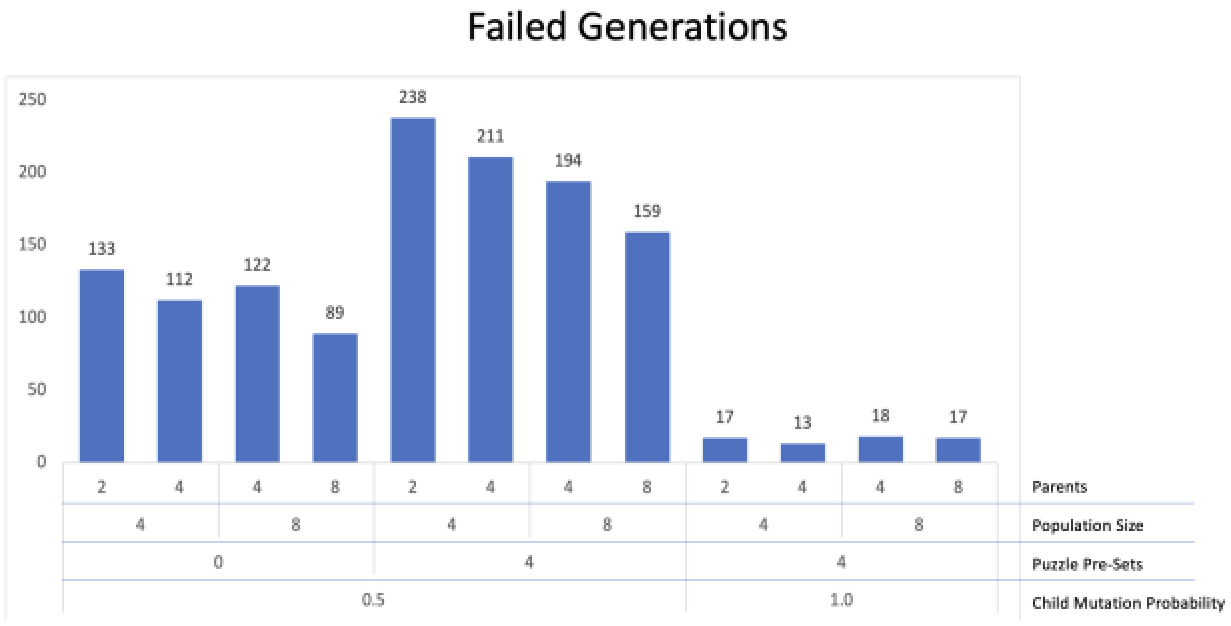


Figure 15. Analysis of failed generations.

Figure 15 shows that failed generations occur more frequently in sets that were run with a CMP of 0.5 than in the sets with a CMP of 1.0

Secondly, failed generations are more frequent in sets ran with puzzles that had four values pre-set, followed by sets ran with blank puzzles. No failed generations were reported for puzzles that had seven pre-set values.

Thirdly, failed generations were generally more frequent in sets where the population size was four.

And lastly, in sets where the population size and the number of parents were not equal there was also a higher instance of failed generations.

The highest incidence of failed generations is in the sets with only two parents in a population of four, using a puzzle with four pre-set values and a CMP of 0.5

- ***Parent Selection Strategy Performance***

The performance of the three selection methods is measured by comparing the following values:

- Average GtC
- Median GtC
- Most frequent GtC
- Frequency of most frequent GtC

See Table 6 for the comparison metrics, Figure 16 for the Ranked Selection frequency distribution, Figure 17 for the Roulette Selection frequency distribution and Figure 18 for the Tournament Selection frequency distribution.

Selection Strategy	Min	Max	Average	Median	Most Frequent	Frequency
Ranked	0	1000	335.311	12	5	859
Roulette	0	1000	176.735	8	4	1435
Tournament	0	1000	362.925	11	5	1018

Table 6. Selection strategy comparison.

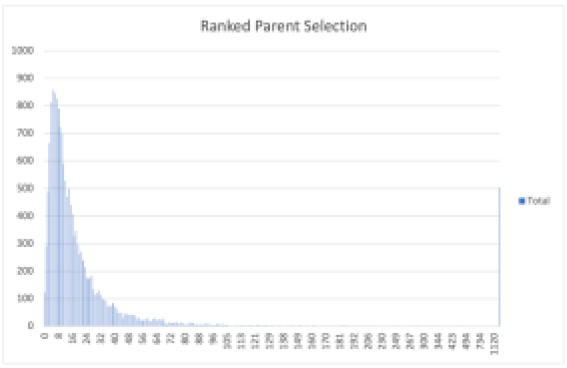


Figure 16. Ranked Selection.

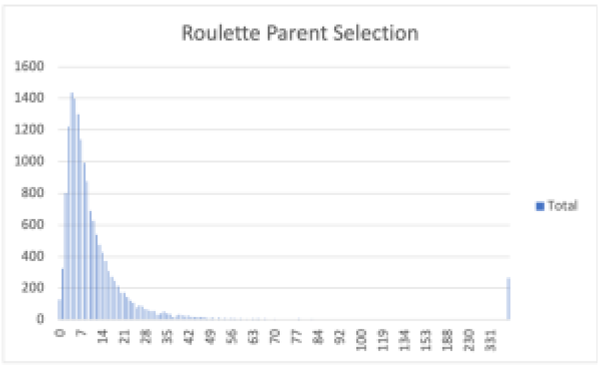


Figure 17. Roulette Selection

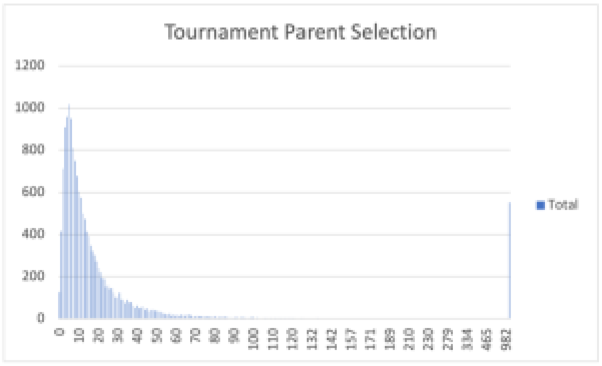


Figure 18. Tournament Selection.

Various attributes of the GtC metric indicate the performance of a parent selection strategy. Firstly, the earlier and higher the frequency distribution peaks, the better the selection method performs. Lower values for GtC are better than higher ones and the more consistently and frequently the selection method produces low GtC figures the better. Lower mean and median values also indicate performance.

Table 7 displays the performance of the selection strategies, Roulette outperforms Tournament, which in turn outperforms Ranked.

Selection Strategy	Rank	Mean	Median	Most Frequent	Frequency
Roulette	1	176.735	8	4	1435
Tournament	2	362.925	11	5	1018
Ranked	3	335.311	12	5	859

Table 7. Selection Strategies ranked by performance.

- ***Analysis***

The performance of Roulette and Tournament selection methods are close, more numerous populations and a larger puzzle structure would improve the strength of this assertion.

This solution is prone to failed generations. These failed generations correlate with CMP, puzzle pre-set values, and the 4x4 structure of the problem reduces crossover points, making gene transfer less effective. Combining with an unfavourable pre-set for a puzzle and a low CMP the likelihood of harmful mutations and subsequently failed populations increase.

4. Future work

This section details suggestions for continuation/augmentation of this project to both improve performance and efficiency.

- ***Optimisation of mutation***

Test results indicate that in general, Almedius converges faster with higher mutation probabilities: as the game restrictions and chosen algorithm result in only two crossover points, variability from crossovers is restricted and mutations compensate for this. Nevertheless, some parameter settings and initial configurations show that a 100% mutation rate may also prevent faster convergence (example see figure 7, where from generation 1 to 2 the overall population fitness did not improve).

For a traditional 9x9 Sudoku, Sato and Inoue (2010) describe a strategy which creates more than two offspring from a set of parents, with different mutations applied, and only the chromosomes with highest fitness surviving eventually. (Deng & Li, 2013) suggest dynamic adaptation of the mutation rate. Both strategies could be implemented and compared to the current implementation.

- ***Parent selection***

Observations indicate close fitness values from the beginning, for which roulette parent selection results in nearly random choices. Therefore, ranked and tournament selection should perform better, if crossovers from fit parents would generally result in fitter offspring.

Our test results show a different behaviour, with roulette being the most effective parent selection method. Further analysis would be required to understand whether this is a result of the specific game, crossover and mutation algorithms chosen, or just caused by the current Almedius design of ranked and tournament selection.

For ranked selection, one area to investigate is the Python enumeration used for weights: the current implementation, for example, assigns a weight of zero the least fit chromosome, which is consequently never chosen as a parent.

Regarding tournament selection, a broad range of design options may be adopted. The current implementation randomly selects half of the population and chooses the fittest chromosome with probability 1.0. Again, this excludes many chromosomes from ever being chosen, thus potentially decreasing variance in the population prematurely.

- ***Redundant configurations derived for edge fitness***

In special cases, the algorithm deriving configuration variants with so called "edge fitness" creates redundant configurations (figure 19). Our measurements confirmed faster convergence with larger populations, thus a single population but with a higher amount of chromosomes would be more efficient. Unnecessary competition between populations with the same initial configuration should be avoided, and therefore the Almedius algorithm be optimised further.

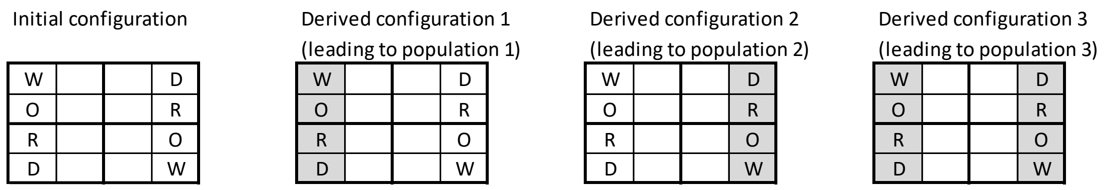

Initial configuration
W D O R R O D W
Derived configuration 1 (leading to population 1)
W D O R R O D W
Derived configuration 2 (leading to population 2)
W D O R R O D W
Derived configuration 3 (leading to population 3)
W D O R R O D W

Figure 19. Algorithm creating redundant configurations from special initial configuration.

5. Conclusion

(Bala, 2017), (Sato & Inoue, 2010) and (Shukla, et al., 2015) applied GAs to a 9x9 puzzle whilst the problem defined in the assignment excerpt was 4x4 which presented various algorithmic manoeuvring challenges. The authors of this report contend that there are so many trade-offs, clear objectives with regards to further optimisation should be set; whether it is avoidance of non-convergence, time required to converge, generations required, and if parallel computing can be leveraged drives design decisions.

References:

Sato, Y. & Inoue, H., 2010. *Solving Sudoku with genetic operations that preserve building blocks.* Copenhagen, Proceedings of the 2010 IEEE Conference on Computational Intelligence and Games.

Bala, A., 2017. A Review of Selection strategies in Genetic Algorithm. *International Journal of Advance Research in Computer Science and Management Studies,* 5(6), pp. 133-141.

Negnevitsky, M., 2011. Evolutionary computation. In: *Artificial Intelligence A Guide to Intelligent Systems.* Dorchester: Pearson Education Limited, pp. 219-257.

Kramer, O., 2017. *Genetic Algorithm Essentials.* Oldenburg: Springer.

Shukla, A., Pandey, H. M. & Mehrotra, D., 2015. *Comparative Review of Selection Techniques in Genetic Algorithm.* s.l., IEEE International Conference on Futuristic Trends in Comupational Analysis and Knowledge Management.

Deng, X. Q. & Li, Y. D., 2013. A novel hybrid genetic algorithm for solving Sudoku puzzles. *Optimization Letters,* 7(2), pp. 241-257.