**FIRST 16 PAGES ONLY**

A dissertation submitted to Manchester Metropolitan University, in part fulfilment of the requirements for the degree of Master of Science in Computing

Viran Mahendra Gohil BA Economics. MSc Economics.

School of Computing, Mathematics and Digital Technology

Submitted April 2017

**ABSTRACT**

This paper presents a genetic algorithm approach to generate strong backgammon automata players. A java program was developed that could generate all legal moves that could be played. A population of individuals with genes represented by numeric weights was initialised to play all other members of the population using the weights to select optimal moves from expert recommended strategies.

The best player in the population was retained unchanged, whilst a tournament was played amongst a randomly selected subset of the rest of the population to identify candidates for crossover and mutation. Optimal individuals identified by running the program with different generations were played against each other, against both a random player and user ‘weight selected’ player. The principal result was the expected strongest player generated by the system demonstrated that it had evolved, although much more work needs to be undertaken to generate a world class player.

No part of this dissertation has been submitted in support of an application for any other degree or qualification at this or any other institute of learning. Apart from those parts of the dissertation containing citations to the work of others and apart from the assistance mentioned in the acknowledgements, this dissertation is my own work.

Signed

**CONTENTS**

**1.** **INTRODUCTION** 6

**2.** **BACKGROUND** 6

**3.** **LITERATURE REVIEW** 8

3.1 TRADITIONAL AI METHODS APPLIED TO BOARD GAMES 9

3.2 BIOLOGICALLY INSPIRED COMPUTING 12

3.3 EVOLUTIONARY COMPUTING 13

3.4 GENETIC ALGORITHMS 14

3.5 GENETIC ALGORITHM PARAMETERS 15

3.6 PARAMETER SELECTION FOR GENETIC ALGORITHMS 17

3.7 WHY GENETIC ALGORITHMS 18

3.8 INDUSTRIAL APPLICATIONS OF GENETIC ALGORITHMS 19

3.9 BIOLOGICAL THEORIES APPLIED TO BOARD GAMES 19

3.10 SOFTWARE DEVELOPMENT METHOLODOLOGIES 22

3.11 OBJECT ORIENTATED ANALYSIS AND DESIGN 25

**4.** **ITERATION - THE GAME** 26

4.1 PROBLEM ANALYSIS FOR THE GAME 26

4.2 DESIGN AND IMPLEMENTATION OF THE GAME 29

**4.2.1 ACTOR AND USER GOALS** 29

**5.** **ITERATION – THE GENETIC ALGORITHM** 33

5.1 PROBLEM ANALYSIS FOR THE GA 33

5.2 DESIGN OF THE GA 34

5.3 GA DESIGN VERSUS THE LITERATURE 36

**6.** **EVALUATION** 38

6.1 TESTING AND ASSESSMENT 38

6.2 CRITICAL EVALUATION 39

**7.** **CONCLUSION** 40

**LIST OF FIGURES**

Figure 1: Flow Chart of Genetic Algorithm for Optimisation 7

Figure 2: Illustration showing the diverse fields of AI. (Oke, 2008.) 9

Figure 3: Chess AI progress compared to human performance. Although written before 1993, the forecast was correct. (Coles, 2002). 11

Figure 4: Waterfall Model 22

Figure 5: Agile Model 23

Figure 6: Agile Iteration 24

Figure 7: Conceptual Object Model. 29

Figure 8: Initial Class Diagrams for the Game 30

Figure 9: Activity Diagram for selecting 1 Move. 31

Figure 10: Backgammon Class Diagrams 32

Figure 11: GA Class Diagrams 35

Table 1: Simple Genetic Algorithm 15

Table 2: Functional Requirements for the System 27

Table 3: Functional Requirements for the Game 28

Table 4: Use Case for 1 Move 29

Table 5: Functional Requirements for the Genetic Algorithm 33

Table 6: Use Case for Genetic Algorithm 34

Table 7: Board Score Computation 37

Table 8: Results 38

**LIST OF APPENDICES**

Appendix A: Software Disc

# **INTRODUCTION**

The aim of this work is to apply one of the search methodologies within the field of artificial intelligence to a backgammon program. Specifically this will be a Genetic Algorithm to evaluate whether a strong backgammon player can be evolved from an initial population of gene-weighted chromosomes or individuals. An improved aptitude for playing the game over generations would demonstrate agent learning.

Beyond this research, with enough resource, a software product could be developed which enables flexibility in selecting population and operator type enabling a valuable addition to the academic literature regards the evaluation of different parameter types. The software could be adapted to other problem domains and/or a retail backgammon product could also be created using calibrated individuals over different generation spans that could be selected via a GUI for humans to play.

However, within the current resource constraints, the following objectives will be addressed;

* Document the academic literature regards the application of AI tools to board games and categorise by tools and trends.
* Document the academic literature regards Genetic Algorithms applied to board games and real world problems.
* Identify and justify the selection of GA parameters and construct, test and evaluate the output of a backgammon program combined with a genetic algorithm.

The following sections in this paper are organized as follows. First, the background motivation and context of the work, a literature review, a detailed analysis of the problem, design of the experiment and software and implementation details, followed by an evaluation of findings, conclusion and recommendations for further work.

# **BACKGROUND**

USD 40 trillion annual economic value to be created by Artificial Intelligence and Robotics by 2025. McKinsey estimates that the annual economic impact of AI and Robotics by 2025 will be USD 5 to USD 7 trillion for automation of knowledge work; USD 2 to 6 trillion for Internet of things; USD 2 to USD 5 trillion for advanced robotics; USD 0.2 to USD 2 trillion for autonomous and near autonomous vehicles and USD 0.2 to USD 6 trillion for 3D printing (McKinsey, 2013).

Although general intelligence is a key goal within AI’s research, it is clear that the interim solutions identified by AI researchers are immediately impacting the economy. Surveying the main AI books (Russell and Norvig, 2003; Luger and Stubblefield, 2004; Poole et al., 1998; Nilsson, 1998) that interim focus is on solving problems around knowledge, planning, reasoning and learning, “….with intelligence exhibited by thinking, making decisions, solving problems, more importantly by *learning*,” (Robin, 2010: 1) which is the objective of this investigation.

Historically, AI researchers did not possess the tools to tackle complex real-world problems. Instead, they used games as the problem domain because, unlike the real world, games have clear objectives and clear boundaries. “By working in these domains, researchers made enormous progress in search, learning, and simulation techniques, to the point where the best computers now surpass the best humans in virtually all classic board games. As a result, AI is now moving on to tackle real-world ambiguity head-on.” (Tesauro, 2013: 1.)

The output of this research will not solve a problem that has been identified in the literature but will be within the trend of AI research whilst serving the author as an introduction to the field and opportunity to create a software program.

In terms of technical details, this is covered in later work, however at this stage, the reader is referred to figure 1. A Genetic Algorithm (GA) is a search-based optimization technique that starts with a population of possible solutions which are assigned a fitness value based on an objective function and evolves individuals in the population using operators such as selection, crossover and mutation of individuals to produce ‘optimal’ children solutions, with the process repeated over a pre-selected number of generations.

Optimal individual solutions have retained the optimal solution within their genes. This solution may have been present in the original population or it has been *learned* via the genetic operators.

Key studies that justify this work are discussed in the following literature review. They include Berliner (1977, 1979), Tesauro (1990, 1995), Pollack (1997), Kotic and Kalita (2003), Koza and Polu (2005), Azair and Sipper (2005a, 2005b), Lucas and Runarsson (2008) and Poli et al. (2008).

Global or Satisfactory Solution

YES

YES

Figure 1: Flow Chart of Genetic Algorithm for Optimisation

# **LITERATURE REVIEW**

The following literature review starts with a brief high-level snapshot of current AI research. It the documents key studies in the history of AI applied to board games where researchers used non-biological methods, such as brute force, alpha beta pruning and evaluation functions. It then introduces biologically motivated computing, its history, methodologies such as genetic algorithms, neural networks and machine learning; the real world value of GA’s and how they have been applied to games including backgammon.

The review is organised in this manner as its starts with a broad overview of AI and then focuses on the evolution of AI tools as they have been applied to board games categorising brute force and alpha beta pruning separately to biologically inspired tools and clearly justifying the selection of the genetic algorithm approach in this research.

AI research can currently be categorised (see figure 2) into the following areas: “reasoning, data mining, distributed programming, artificial life, expert systems, genetic algorithms, systems, knowledge representation, machine learning, natural language understanding, neural networks, theorem proving, constraint satisfaction, and theory of computation” (Oke, 2008: 1).

Board games are a valuable arena for research as they have closed state spaces with well-defined rules but are not superficial since a comprehensive search of the state space is often so challenging that new methods must be conceived. Thus, they present the right level of test for researchers.

AI Applications

Figure 2: Illustration showing the diverse fields of AI. (Oke, 2008.)

## 3.1 TRADITIONAL AI METHODS APPLIED TO BOARD GAMES

Arthur Samuel (Samuel, 1959; 1967) undertook pioneering work programming a computer to play checkers. He used a low ply mimimax search combined with a heuristic function to guide evaluation. He created the first AI that was able to learn its own evaluation function by playing predecessor functions, creating an above-average beginner player. Parts of Samuel’s research were analogous to temporal-difference learning as he implemented a learning method that included temporal difference ideas, an approach successfully executed by Tesauro (1995) for Backgammon. There is a discrepancy in the published literature, with Johns (2009) reporting that Samuel created a better-than-average player, whilst McCarthy (1990) reports that Samuel’s Checkers program beat the 4th rank player in the US. Fogel (2001) was unable to verify this claim.

Nevertheless, Samuel’s work is commended for its contribution to the field and for its commercial value influencing IBM in computer design (McCarthy, 1990). However, it was not until 1992 that a substantial expansion in Samuel’s work could be made, with the development of Chinook, (Schaeffer et al., 1992; 1993) which used a database of 444 billion (8-piece) end-move configurations combined with alpha-beta search. In 1994, Chinook became world champion. Further work on checkers was undertaken by Chellapilla and Fogel (1999), who developed the use of an evolutionary algorithm to evolve the weights of a neural network. The resulting player won six out of six games versus a commercial checkers program.

Checkers has a branching factor of 10, whilst chess has a branching factor of 35. A computer would need to research 35 ^ 80 plies moves in an average game. For Simon (1992), choosing chess as an environment for the application of new algorithms was an institutionalized landscape for testing new AI methods. “It was both simple enough to be able to formalize mathematically and yet complicated enough to be theoretically interesting.” (Newell et al., 1958: 326.) Wiener (1948) first published a theoretical description of how a depth limited minimax search combined with an evaluation function could be utilised to develop a chess program. Shannon (1950) published an outstanding paper coming to the same conclusions and advancing further to build a chess automaton not capable of playing a full game. Turing wrote the first Chess program, in 1952, but could not complete it. In practice, the theoretically computable tree proposed by Shannon and Wiener was unachievable, due to the branching factor, although Shannon recognised this and proposed a reduced look ahead and the addition of a heuristic to discount suboptimal moves. Shannon had proposed what McCarthy later called an ‘approximation’ (McCarthy, 2006) of Alpha Beta pruning.

With the advent and development of alpha-beta pruning ‘originally’ proposed by McCarthy at the Dartmouth conference in 1955 (Berstein and Roberts, 1958), the simplicity of Shannon’s minimax procedure and alpha beta slowly permeated through an academic community that practically discounted other approaches (Marsland et al., 1991). Thus, the main focus was applying pruned computational speed brute force mimimax search. The first full programs that could play chess credibly at amateur level using such a method were created by Kotok (1962), a student of John McCarthy at MIT.

The first computer to beat a grandmaster was a computer wholly dedicated to playing Chess. Belle, developed at Bell Labs, became the first master level machine in 1983. Thereafter, the Hitech program (Berliner, 1989) dominated computer chess from 1985 to 1988 and was the first computer to win a match against a Grand Master, in 1989. By this time, researchers had concluded that the pursuit of killer heuristic improvement was slow and developing human knowledge in the evaluation function only generated middling players (Berliner, 1989), whilst increases in computer search capability added the most to tournament rankings. It was not until 1996 that a world champion, Gary Kasparov, lost a game to a computer known as Deep Blue. The reader is referred to Coles (2002) for a detailed consideration on the history of chess automation.

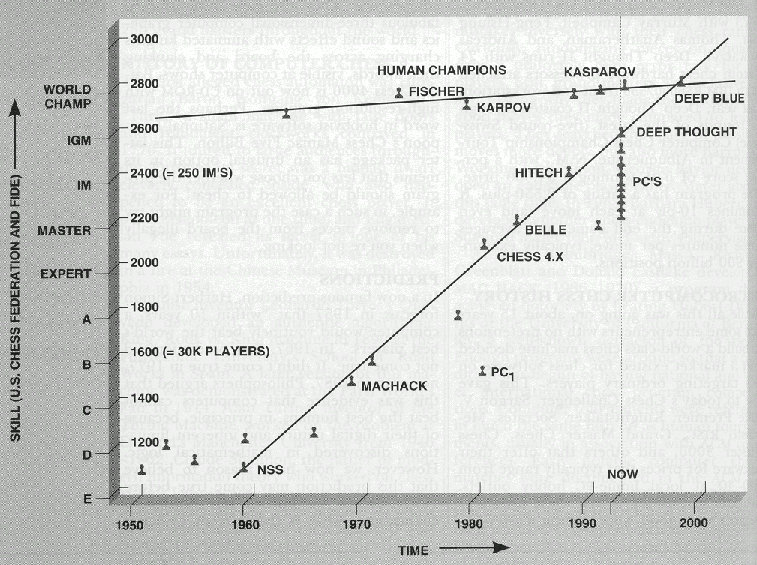


Figure 3: Chess AI progress compared to human performance. Although written before 1993, the forecast was correct. (Coles, 2002).

Although Deep Blue’s win was a shock to the world of chess, and momentous in representing the success of hardware and computing power, many academics questioned the win as brittle due to the parallel custom-designed hardware and bespoke fine tuning to address Kasparov’s game (Ensmenger, 2012). Deep Blue’s intelligence is disputed because the basis of that intelligence was a bespoke brute force search. Deep Blue typically searched between 6 and 16 ply but up to 20 ply in certain situations (Aleksander, 2001). Noam Chomsky has criticized this aspect of game-playing research as being “about as interesting as the fact that a bulldozer can lift more than some weight lifter” (Chomsky, 1992: 189). This raises the question of what is intelligence. Do grandmasters and human beings search for possibilities within the human brain without cognition and then intuitively recognise an optimal answer along the lines of deep blue brute force search or is intelligence a more thoughtful heuristic evaluation. The generally accepted conclusion is that Deep Blue is very good at playing chess as modern hardware has made brute force search more practical (Aleksander, 2001). With further computing hardware power, modern day chess engines only need a limited evaluation function and pruning rules to achieve the equivalent performance of the best human players.

The deep blue conclusion to chess research is unsatisfactory and contradictory to the objectives of AI (Ensmenger, 2012). Although Chess was a good sandbox to test algorithms, an additional reason that it was originally selected for AI research was that it held a particular fascination in society, as an aptitude for the game was considered the peak of human intelligence. The argument here is that if machines could replicate chess intelligence, then it would be a satisfactory and significant contribution to long-term general artificial intelligence. However, researchers may have been “led astray,” as it were, from this real goal of AI, as the goal became chess-only rather than AI. This dominance of chess in approaches to AI research, with particular focus on trees and mimimax, is now questioned by many academics (Coles, 2002; Ensmenger, 2012) as having overshadowed other problem domains and techniques. It has distracted researchers from more generalisable and theoretically productive avenues of AI research (Marsland et al., 1991) and chess ultimately produced little in terms of fundamental theoretical insights.

Thus, computational AI problems require a search through many possibilities to find a solution. Conservative methods, as brute force, are simple to apply and yield reasonable algorithms; however, in many cases, the main disadvantages of state space size or large branching factor leads to an unacceptably slow or impossible search, hence why such tools as pruning and evaluation functions were added by researchers (Kotok, 1962). However, with brute force and those additional tools, it is practically impossible to model stochastic problems which mimic real-world uncertainty or situations with very high branching, hence the criticism of tournament winning approaches to chess as distractions from creating tools to solve real-world problems.

Further methods that can combine with and counter brute force’s disadvantages are Goal Search, (Harrison, 2010) where the target is nodes that complete short-term tasks; Monte-Carlo evaluation functions, which evaluate random games played from the immediate state (Chaslot et al., 2006); supervised machine learning, where the program learns the function that has previously generated the desired output from a set of inputs; reinforcement learning, where the program receives a signal or ‘reward’ that it is moving in the right direction; neural networks, which learns by updating the mappings between facets of games states; and outputs and evolutionary methods.

## 3.2 BIOLOGICALLY INSPIRED COMPUTING

Up to Deep Blue’s victory, we have seen that high-branching factor has been combated with brute force and how this has been perceived negatively by some researchers and enthusiasts for its lack of contribution to the aim of general intelligence. Clearly, the forefathers of computer science—Turing, Von Neumann, Weiner, and others—were driven by dreams of ingraining computers with the human-like intelligence, whilst the overuse of brute force search is disparate to this.

This brings us to biologically motivated computing endeavours. A bit-part player that has seen a resurgence in the last 30 years, evolving into the areas of neural networks, machine learning and evolutionary computing.

Machine learning identifies [algorithms](https://en.wikipedia.org/wiki/Algorithm" \o "Algorithm) that can [learn](https://en.wikipedia.org/wiki/Learning" \o "Learning) from and make predictions on [data](https://en.wikipedia.org/wiki/Data" \o "Data).  This can be in the form of ‘[supervised learning](https://en.wikipedia.org/wiki/Supervised_learning" \o "Supervised learning),’ where the computer detects patterns from historical inputs and outputs and ‘[unsupervised learning](https://en.wikipedia.org/wiki/Unsupervised_learning" \o "Unsupervised learning),’ wherein no output or no input (to learn features) is given to the computer or [reinforcement learning](https://en.wikipedia.org/wiki/Reinforcement_learning" \o "Reinforcement learning) where a computer receives a signal in a dynamic environment that it is getting closer to the desire output.

Artificial neural networks are a form of machine learning conceived by Warren McCulloch (McCulloch and Pitts, 1943). They consist of interdependent processing components (neurones) working in conjunction, mimicking human learning by example. Neural networks have an exceptional aptitude to detect patterns in complex data and provide the commensurate forecast. Specifically, they are very good at identifying associations or regularities in high volume and/or vague date where conventional techniques are inadequate. However, it is this independent problem solving ability, particularly in the case of backpropagation, that makes them unpredictable owing to the opaqueness of their analysis. Researchers have to accept the forecast as given. A detailed treatment is offered by Hassoun (1995).

## 3.3 EVOLUTIONARY COMPUTING

Evolutionary computing is an abstraction of the theory of biological evolution. The central argument is that evolution has optimized biological processes, since evolution in itself is a biological process; therefore, evolution must have optimised itself (Rechenberg, 1989). Thus, adoption of the evolutionary paradigm to computation and other problems mimics the reality of discovering optimal solutions.

Although there are many different types of evolutionary algorithm, all with their own distinct lineages in the research—such as ‘evolutionary strategies,’ (Rechenberg, 1965; Schwefel, 1965) which uses non-string representation, randomly selects parents and only applies a mutation operator; and ‘evolutionary programming,’ which only applies mutation—the author will focus on the most popular subset of evolutionary programming (Fogel, Owens, and Walsh, 1966), which is ‘genetic algorithms,’ (Holland, 1975) whilst identifying some key studies in the overall history of the category. See Hoffmeister and Back (2005) for a comparison of evolutionary approaches.

The first record of non-biological evolution was published by Barricelli (2006), in 1954. He simulated numbers in a grid that moved in the grid according to rules for each number. From simple moves rules, he generated patterns dubbed as “organisms.” He noted autonomous evolution and recombination (also known as crossover), including newly formed offspring of colliding patterns demonstrating multi-point crossover.

In an exemplary piece of work, Baricelli et al. (1967) investigated a two-player card game, wherein cards could be ascribed either high or low value. Possible bets included pass costing 2, low costing 3 and high costing 7 units. If all players betted the same in this game, then the card of highest value would win; however, had they betted differently, the participant with the highest bit would win. The individual representation composed of four numbers for betting probabilities under different outcomes and four numbers for genetic parameters, including mutation, generation and crossover. Hence, the individual was encoded with both the roles to bet and the method for searching for new ways to bet, including recombination. In contemporary nomenclature, this process is referred to as “self-adaptation.”

Conrad (1985) developed the first known simulation of a hierarchical and potentially open-ended ecosystem, a population of individuals subjected to a strict *materials preservation rule*, activating a clash for survival. The organisms could collaborate as well as initiate genetic crossover and the mutation of the weights of their genome. Without a fitness function, the program was an ecosystem in which genetic operations would instigate different behaviour patterns. Conrad witnessed the evolution and coexistence of two breeds, resistant to gene changes and identified as descendants of clans with distinct survival tactics. Of particular interest was the appearance of attributes favouring the survival of the individual versus reproduction, demonstrating that individuals can evolve to a state that is disadvantageous to evolution. Conrad’s work was a clear forerunner to the later work of Ray (1992), wherein assembly language programs competed for CPU cycles. Conrad is also credited with the first modern effort to use an evolutionary algorithm to optimize a neural network (Conrad and Kampfner, 1983).

Fraser (1957) was a pioneer of genetic optimisation originally investigating simulations involving diploid organisms, where he used binary genes in the chromosome and a crossover probability at gene level. His key contributions to the area included the development of a variety of tools, including Monte Carlo probability sampling, and studying the effects of linkage, epistasis, genetic variation and evolutionary speed. (Fraser, 1957; 1965; 1967.) Fraser inspired researchers’ modelling, which now include operators for crossover and mutation.

In retrospect, Fraser’s work was ignored outside of actual biological genetic experimentation, although his computational procedures were a precursor to the methods that subsequently become standard in GA applications, as Fraser effectively uncovered the same concepts as Holland (1975; Fogel, Undated).

Due to the previously disconnected approach and overt focus on the mutation operator, GAs only rose to prominence with John Holland’s (1975) work in the 1960s and ’70s. Holland is considered the patriarch of the Genetic Algorithm, as he introduced the mating and crossover operators and proposed a general way to define adaptive systems (Holland: 1962, 1962a, 1969). Holland inspired his Ph.D. students to research diverse areas, such as chemical and molecular genetics (Rosenberg, 1967); variations of adaptive plans, called “reproductive plans” (Cavicchio, 1970); genetic algorithms for search (Hollstien 1971; Frantz, 1972); and continuous function optimization (Hollstien, 1971; Jong, 1975). Holland wrote the seminal book formalising adaptive systems, specific tools and capabilities.

## 3.4 GENETIC ALGORITHMS

The main facets of genetic algorithms are a population of individuals or chromosomes with strings of binary genes representing candidates’ solutions. A positive value, generally known as fitness value, is used to reflect the degree of “goodness” of the chromosome for solving the problem. In each cycle of genetic operation, a ‘selection operator’ chooses chromosomes that will be allowed to reproduce. A ‘crossover operator’ exchanges parts of the two successful chromosomes and a ‘mutation operator’ randomly changes the genes solutions.

|  |
| --- |
| The Simple Genetic Algorithm  Step 1. Define a suitable representation of the problem to be solved.  Step 2. Create an initial population of *N* individuals for evolution.  Step 3. Define a suitable fitness function for evaluating the individuals.  Step 4. Perform genetic operations (crossover and mutation) to generate possible offspring.  Step 5. Evaluate the fitness value of each individual.  Step 6. Select superior *N*individuals according to their fitness values.  Step 7. If the termination criterion is not satisfied, go to Step 4; otherwise, stop the algorithm. |

Table 1: Simple Genetic Algorithm

## 3.5 GENETIC ALGORITHM PARAMETERS

Referring to Table 1, regarding step 1, there are different ways to encode the individual, which can impact the prospect of attaining a solution and quality of solution. Known as the “representation problem,” this is a key avenue of research within genetic algorithms, especially if one were to consider, say, job-shop scheduling, known to contain the most challenging combinatorial optimization problems, hence an area where representation has been widely investigated (Werner, 2013). The two key areas of research to improve genetic algorithm approaches have been focused on identifying heuristics and the best ways of encoding the representation. (Muh-Cherng et al., 2017.)

The most common representation is binary (Tsai et al., 2015), wherein the genotype consists of bit strings, although the weakness of binary encoding lies in the fact that different bits have different significance, which can ‘mute’ the impact of mutation and crossover operators. Other types are: ‘real-valued representation,’ in which the desire is to represent the genes as continuous rather than discrete; ‘integer representation,’ where the desire is not to code in a binary ‘yes’ or ‘no’ format; and ‘permutation representation,’ relevant when an ordering is required in the solution, as in the case of the travelling salesman.

The value of binary encoding lies within the granularity of the solution, as the conversion of integers to bits, as well as creating more data with which to work, create, to a modest degree, ‘cleaner’ mutations in terms of flipping, although the operators may have a somewhat muted effect in the generation of variety in the gene pool, thereby increasing the requirement for more of both generations and computing runtime. One method using real or integer numbers while countering this lack of granularity is differential evolution, adopting a modified mutation and crossover operator that makes use of the difference between multiple integers and real vectors to create a new vector, by adding both a random proportion of the difference and a random noise (Storn, 2008).

In regard to step 2, the two population models widely used are both randomly generated, the first being generational and the second steady state. The generational has *N* offspring, wherein *N* represents the population size, the entire population being replaced by the new population after mutation and crossover. Steady state only applies mutations and crossovers to a select few individuals and integrates them to the original population.

One major problem with GAs is premature convergence to local optima. Normally occurring owing to the phenomenon of high-fitness individuals dominating the population, thus constraining the operators from evolving higher fitness children, diversity in the population is, thus, paramount to achieving global optima. Conventional GAs tend to model the population as a both simple and non-ordered set of individuals, allowing any persons to crossover. Unconstrained mating in a finite population, however, may lead to genetic drift, thereby diminishing individual diversity. Methods to counteract such a phenomenon include organising the population space and introducing heterogeneity. In organising the population space, researchers can include either spatial segregation (Artyushkenko, 2009; Da Silva and Simoni, 2001; Ursem, 1999) or spatial distance (Alba et al., 2005), thereby resembling real-world geographical separation. Heterogeneity can manifest in the form of sex or gender difference, age (Kubota and Fukuda, 1997) or religion (Thomsen et al., 2000) or any number of attributes or preferences, although the final form of heuristic individual initialisation just mentioned can result in a population with little diversity, which can be countermanded by introducing some random solutions alongside.

In regards to step 3 in Table 1, the fitness function is an objective function that determines the optimality of a solution. It is relevant as diversity of the fitness function can counteract the representation problem, as the diversity in the fitness function can increase the search space. The most common method of fitness selection (the selection operator) is based on *elitist selection*, under which the fittest individuals are chosen. Others methods include *roulette-wheel selection*, whereby the probability of individual selection is dependent on the level of fitness versus other individual’s; *tournament selection*, where members of subdivided populations compete; linear rank selection, where selection is based on rank order of fitness, exponential weighted selection; and truncated selection which is the equivalent of selected breeding.

The negatives of elite and roulette wheel selection are their reliance on fitness numbers, which may identify robust but imperfect individuals. To counter this, the rank operator selects using proportional rank versus other individuals.