Computational Intelligence in Stochastic Solution for Toroidal N-Queen

A.A. Ojugo and R.E Yoro
[1]*Department of Mathematics/Computer Science, Federal University of Petroleum Resources Effurun, Delta State, ojugo_arnold@yahoo.com, arnoldojugo@yahoo.com*
[2]*Department of Computer Science, Delta State Polytechnic Ogwashi-Uku, Delta State rumerisky@yahoo.com*

Abstract

Evolutionary models are now robust tool in modeling complex, dynamic and non-linear data processing due to its flexible mathematical structure to yield optimal results even with imprecise, ambiguity and noise at its input. The study investigates evolutionary models for solving Toroidal N-Queen task. Evolutionary hybrids have become veritable models for multipoint search in CSPs with application areas like image/video analysis, military, network construction, OS resource scheduling and allocation, medicine, cloud and clustering computing etc. Solution space representation and fitness functions (as common to all algorithms) are discussed. For the adopted support/confidence model, $\varpi 1=0.2$ and $\varpi 2=0.8$ respectively yields better convergence. Other suggested value combinations led to a slower or non-convergence. For 20-queen, HPSOGA and CGA found an optimal solution in 32 seconds after 188 iterations; GSAGA found its optimal solution in 18seconds after 402 iterations and consequently, GASA found an optimal solution 2.112seconds after 391 iterations respectively.

Keywords: *Swarm, Agents, Particles, Elitist, Constraints, Fitness Function*

1. Introduction

Soft Computing (SC) harness Artificial Intelligence, to solve tasks by exploiting knowledge as symbolic reasoning translated into models that are tolerant to noise, imprecision, uncertainty and partial truth in the data. Its quantitative data processing ensures qualitative knowledge and experience via optimization methods such as genetic algorithm, neural network etc [1]. SC mimics natural agents seeking survival and have proven efficient and optimization displays 3-feats in its attempt to explore dynamism: *robustness*, *adaptation* and *flexibility* [28]. Thus, they convert into mathematics, biological processes in the fastest time to yield implicit, predictive models that stems from experience in its ability to recognize data feats and behaviours via an optimal fitness of high quality, void of overfitting that constantly affects any solution's quality [4].

1.1. Toroidal N-Queen Overview

This is a classical CSP with variable permutation that yields an optimal solution assigned unique values – to satisfy some constraints by placing on *n x n* chessboard, N-queens so that on each row, diagonal, column and extended diagonal – there exists only one queen where no queen attacks another as in fig. 1. There are 92-solutions for the 8 X 8 N-Queen task; out-of-which there exists 12-fundamental solutions (none, a reflection/rotation of another). This is today found in introductory artificial intelligence as benchmark in CSP [11,12,16].

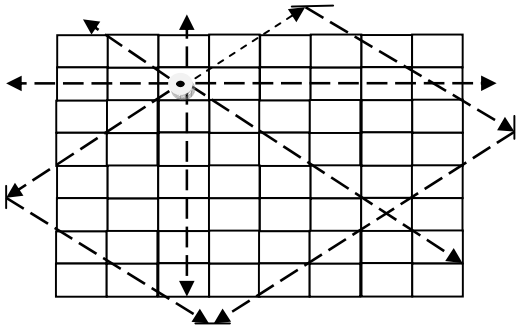

Figure 1. Toroidal 8 X 8 Queen solution

A motivation of this study is the dynamic and difficult nature of its solution. Stochastic method finds global optima in multipoint task, where many local optima (solution that satisfy some constraints) exists, with systemic search on encoded (continuous or discrete) space whose solution via hill-climbing method – may gets stuck at local minima due to their speed. Thus, the space is searched until global optima are found [6, 25, 30].

The Toroidal N-queen notes: how many ways can a queen be placed on the *n x n* chessboard so that no two queens are on same row, column and extended diagonal as in fig. 2, first posed by Poyla – who showed T(n) > 0, if and only if (n,6) = 1 [2]. This study explores the implementation of various stochastic evolutionary optimization methods. Each is implemented and tested on figure 2.

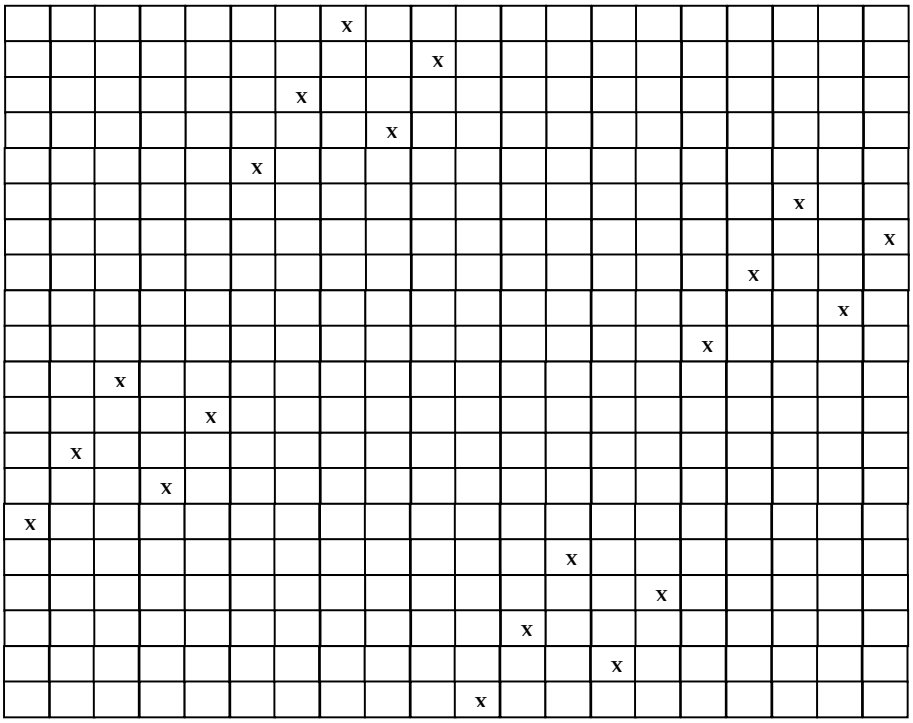

Figure 2. A 20-Queen Toroidal Queen

1.2. Solution Space Representation

Fig. 1 has 64-empty cells representing the solution space, using any of both approaches namely:

a. First method, each of the 64 empty cells is a separate agent that requires its own population. Thus, the solution space consists of 64 separate population groups. A demerit of this approach is that each agent is operated upon separately, preventing the possibility of interaction between such agents – making it computationally, more challenging.
b. Second, teat as combination the 64-empty cells with only 8-cells marked by an agent each. Thus, the solution space has only one population; instead of 64-different solution groups as first case. This allows greater interaction in agents and is computationally, less demanding.

1.3. Fitness Function

The only possibility is to implement a good fitness function that determine (as represent queens via integers) if another queen is repeated or not present in row, column, diagonal or extended diagonal. Fitness value is assigned a possible solution based on number of repeated or non-present integers. The more the non-present or repeated integers there are in the solution's row, column, diagonal and ex-

tended diagonal, the higher the fitness value assigned to that solution. Thus, only repetitions in rows, columns, diagonals and extended diagonals are considered. These, contributes to the fitness value [26].

1.4. Statement of Problem

Studies have shown most of the evolutionary algorithms that employs backtracking for local, systemic and global search in discrete and continuous state spaces use hill-climbing method that often get them stuck at local minima (a function of their speed). Thus, hybrids are designed to cub such defects.

1.5. Objective and Significance of Study

The study explores Toroidal Queen using the second solution space via: (a) Particle Swarm Genetic Algorithm, (b) Cultural Genetic Algorithm (CGA), (c) Gravitational Search Algorithm Genetic Algorithm, and (d) Genetic Algorithm Simulated Annealing respectively.

Its application yields veritable tool in computational intelligence for dynamic processes and CSP tasks, applied in areas such as image/video analysis, simulation, data routing and compression, network construction, multiprocessor load balancing, OS resource allocation, medicine, finance, military, engine automation, fault diagnosis, forecasting, data mining, cloud and clustering computing etc.

1.6. Limitations of Study

Hybrids are more difficult to implement (though they yield better search space selection). Its encoding may yet not be able to address the general problem of existing statistical dependencies amongst data. This is curbed by encoding via structured learning in crossover, mutation and schedules etc.

2. Hybrid Particle Swarm Optimization-Genetic Algorithm (HPSOGA)

Hybrid Particle Swarm Genetic Algorithm adopts the population based speed of PSO and permutation, parallel search of GA. The background is as thus:

2.1. Particle Swarm Optimization (PSO)

PSO as a population based heuristic, simulates motion via swarm collective intelligence to specify a model of randomly initialized candidates distributed in space towards an optimal solution in a number of moves based on large amount of data about the domain, assimilated and shared by the swarm. To generate and select particles adapted to their environment via a fitness function, constraints that must be satisfied in a number of moves and desirable traits feat evolves but remains in the swarms' composition (solutions generated in that move) – better traits replace weaker ones [17,18,31]. PSO is continuous but modified to handle discrete values, and its steps are as thus:

a. **Position/Velocity Update:** A particle is point in space that changes its position during moves based on updates. Positions Xi and velocities Vi of an initial swarm are randomly generated with lower and upper bounds in Eq. 1 and 2, randomly distributed in space as:

$$X_o^1 = X_{min} + rand(X_{max} - X_{min}) \quad (1)$$

$$V_o^1 = \frac{X_{min} + rand(X_{max} - X_{min})}{\nabla t} = \frac{Position}{Time} \quad (2)$$

[19-20] notes that particle velocity update is done via a fitness function that determines a particle's best as a function of its current position and the swarm's best global value in current swarm (Pgi) and best position of each Pi (current and previous moves). Velocity update uses effect of the current motion (V_i^t) to give a direction V^i_{t+1} for the next move. To avoid local optimum entrapment, it uses uniformly distributed rand() and 3-weights factors namely: (a) inertia ϖ, (b) particle own memory (self confidence) $\phi 1$ and (c) swarm confidence (swarm factor) $\phi 2$ of Eq. 3:

$$V_{t+1}^{1} = \omega + V_{(t)}^{1} + \varphi_1 rand() \frac{(P^1 - X_t^1)}{\nabla t} + \varphi_2 rand() \frac{(P_t^2 - X_t^2)}{\nabla t} \quad (3)$$

V^{i}_{t+1} is particle Velocity i at time t+1,
$\varpi+V^{i}t$ is current motion, $X^{i}t$ is particle position in time t,
$\phi 1$*rand() [(Pi-Xit)/∇t] is particle's influence factors
$\phi 2$*rand() [(Pgt-Xit)/∇t] = Swarm's influence factors

b. **Position Update** is in 2-steps: (a) velocity/position update, and (b) fitness calculation, as repeated until a convergence criterion is reached. Stop criterion is set (max change in best fitness is smaller than needed number of move as in Eq. 4:

$$X_{t+1}^{1} = X_t^1 + V_{t+1}^1 * \nabla t \left| f\left(P_t^g\right) - f\left(P_{t-g}^g\right) \right| \in \varepsilon \quad (4)$$

[14, 21] notes that variables may hold values beyond X_{max} and X_{min}, due to its current position and calculated velocity – as velocity grows rapidly. The particles diverge instead of converge, and they are dragged back to nearest side constraint via Eq. 5 (handles particle velocity explosion and constraints via linear exterior penalty, if such particle violate bound value).

$$f(x) = \varphi(X) + \sum_{i=1}^{N_{max}} X_i * \max[0, g_i(x)] \quad (5)$$

2.2. Genetic Algorithm (GA)

GA is a population optimization inspired by Darwinian evolution and genetics (survival of fittest and natural selection). It consists of a population (set of numeric data) chosen for natural selection that consists of potential solutions to a specific task with each potential solutions referred to as an individual (combination of genes). An optimal combination of genes can lie dormant in the population (from a combination of individuals). An individual with a genetic combination close to the optimal is described as fit [8, 16].

A new pool is created by mating two individuals from the current pool. The fitness function is then applied to determine how close an individual is to the optimal solution. The selection function ensures that the genetic data from the fittest individuals is passed down to the next generation or pool – so that a fitter pool emerges. Thus, the new population or pool will converge to optima or gets close to it as possible. GA is carried out in four steps namely:

a. Initialize – encodes data into a format suitable for selection. Each encodings has its own merit and demerit. Binary encoding needs more bits to do so and its demerit is in its length – making it computationally, more expensive [7]. Decimal encoding allows greater diversity in chromosome and greater variance of pools generated [9]; while floating point encoding allows individual to be encoded as floating point numbers or its combination, and is far more efficient than binary since values encoded are similar with character and commands to represent an individual.

   We encode solution as fixed length candidate vectors in one or more pools of different types. The *fitness* function sees a solution from candidates evaluated, to determine its goodness of fit. If solution is found, function is *good*; else, is *bad* and not selected for crossover. The fitness function is the only part with knowledge of task at hand. The more solutions are found, the higher its fitness value [5, 7]. The Support and Confidence model is as thus:

   If A then B,
   Support = |A and B| / N
   Confidence = |A and B| / |A|
   Fitness = w1 * support + w2 * confidence

b. Selection – First, a fitness function is used to determine how close an individual is to an optimal solution. After which, individuals are selected for mating. The *tournament* method is adopted which selects a random number of the individuals in pool and the fittest individual is selected. The larger the number of individuals selected, better the chances of selecting a fittest individual. This continues until one is chosen, from the last two or three remaining solutions, which become selected parents to create the new offspring. Selection ensures that the fittest individuals are selected and more likely chosen for mating but also allows for less fit individuals from the pool and the fittest to be selected. The *tournament* method is easier and more efficient to code, and best suit for parallel architectures so that selection pressures are easily adjusted [7, 13]. A selection that only mates fittest is *elitist* and often leads local optima. The Tournament selection algorithm is as thus:

Algorithm: Tournament Selection {}
1. Input: Population of chromosome
2. Output: Selected Chromosome for crossover
3. Randomly select 3-chromosomes from pool
4. Pick best 2-solution based on fitness value
5. Return the selected two solution
6. Apply Crossover | Select best solution as parent

c. Crossover – involves the reproductive process in which two individuals exchange their genetic materials to yield a new, fitter individual while ensuring that genes of fit individuals are mixed in an attempt to create a fitter new generation. The various crossover types depend on encoding. For example, (a) simple crossover on binary encoded pool, by choosing a particular (multi) point where all genes are from one parent, and (b) arithmetic crossover in which new pool is created by adding percentages of one individual to another [7, 24].

d. Mutation: A child's chromosome is slightly altered by either changing its genes or its sequence, so as to ensure pool converges to global minimum, instead of local optimum. Algorithm terminates once optimal solution is found. Computationally expensive though, GA also stops when a number of new pools are created or if no better solution is found. A gene may (or not) change depending on mutation rate. Mutation improves the pool's diversity as needed in reproduction [7] and its algorithm is as thus:

Algorithm for Mutation{}
1. Input: A chromosome rule
2. Output: Mutated solution, a fns of mutation rate
3. Set mutation threshold (between 0 and 1)
4. For each network attribute in chromosome
5. Generate a random number between 0 and 1
6. If random number > mutation threshold then
7. Generate Random value for N-Queen
8. Set solution attribute value with
9. Generated attribute value
10. End if
11. End For Each

For a permutation task such as the Toroidal N-Queen – since all elements are dependent of each other, it implies that all conflict must be resolved. Particle velocity is added on each dimension to update the particle – a distinct measure. If velocity is larger, particle explores more distant areas and is more likely to change to a new permutation series – and new velocity in such a permutation scenario signifies possibility of particle change (velocity update formula remains same). However, velocity is limited to absolute values (that is difference between particles) [27].

After this, fitness function determines the fitness of each individual in pool. From which a sub-pool of 30 individuals is selected for reproduction via tournament, to determine mating individuals. Both crossover (single point) and mutation is carried out – in which a number between 1 and 64 is

randomly generated from a Gaussian distribution that corresponds to the crossover point. All genes before this point come from one parent; while the other parent contributes the rest. A new individual, whose genetic makeup is a combination of both parents is thus, reproduced. The new individual also undergoes mutation from which three random genes are selected for mutation and are allocated new random values that still conforms to the belief space. The new individuals replace ones in the pool, with low fitness values (creating a new pool). This continues until an individual with a fitness value of zero (0) is found – to imply that the solution to the puzzle has been reached [3].

Mutation is then applied on each generated pool and number of mutation applied depends on how far the GA has progressed (how fit is the fittest individual in the population). If number of mutations applied equals the fitness of the fittest individual divided by 2. If fittest individual is 17, number of mutation equal 9. Thus, knowledge of solution has direct impact on how the algorithm is implemented. Algorithm terminates when the best individual has a fitness of 0 – solution is found [10].

3. Cultural Genetic Algorithm

Cultural GA is one of the many variants of GA with a belief space noted as: (a) Normative (where there is a particular range of values to which an individual is bound), (b) Domain (data about task domain), (c) Temporal (data about events' space is available) and (d) Spatial (topographical data). In addition, an influence function mediates between belief space and the pool – to ensure and alter individuals in the pool to conform to belief space [10]. CGA is chosen as o yield a pool that does not violate its belief space and reduces number of possible individuals GA generates till an optimum is found [25].

Study uses second solution space (individual consist of 64-genes, corresponding to a non-fixed cell on chessboard). A pool of 64-randomly initialized agents, each containing genes that conforms to the belief space as: (1) Normative (individual has genes ranging from 1-to-20), (2) Domain (individual have integer as genes), (3) Spatial (individuals have integers 1-to-20 exactly once within same row, column or extended diagonal, in fig. 2), and (4) Temporal (mutation must not alter values of fixed cells). The *third* belief has topographic knowledge of the space (i.e. fixed cell values). An influence function ensures the belief space is adhered to. It also has a rounding function to ensure all values are integer and ensures the random numbers generated are not repetitions of one of the fixed numbers in the same row, column and extended diagonal [29].

Once initialized, fitness function computes each individual's fitness, from which 30-individual sub pool is selected for reproduction via tournament, to determine mating individuals. In reproduction, both crossover (single point) and mutation is carried out – in which a number between 1 and 64 is randomly generated from Gaussian distribution, corresponding to the point of crossover. All genes before here, are from one parent; while the other parent contributes the rest. A new individual, whose genetic makeup is combination of both parents is produced, and also undergoes mutation from which 3-random genes are selected for another mutation and are allocated new random values that still conforms to the belief space. New individuals replace ones in pool, with low fitness values (creating a new pool). This continues until individual with a fitness value of 0 is found. Thus, solution is been reached [3].

Mutation is applied on the pool and the number of mutation applied depends on how far CGA is progressed (how fit is the fittest individual in the pool). Thus, number of mutations equals fitness of the fittest individual divided by 2. If fitness of the fittest individual equals 31, number of mutation equals 16. Initialization and selection ensures the first 3-beliefs are met; while mutation ensures the fourth is met. Also, an influence function helps influence how many mutations takes place. Knowledge of solution (how close task is to solution) has direct impact on how algorithm is processed. Algorithm stops when best individual has a fitness of 0 [10, 29].

4. Gravitational Search Algorithm Genetic Algorithm (HGSAGA)

GSAGA is a powerful optimization method, which explores GA's parallel ability to search a space via multiple individual and GSA's speed and flexibility in finding a better optimal point even when a local minimum is found. Both are essential in finding solution to a Toroidal queen [9].

4.1. Gravitational Search Algorithm

GSA is based on laws of gravity/motion of isolated masses, and each mass is a solution space. It states "as particle attracts each other, the gravitational force between them is directly proportional to their masses product and inversely proportional to their distance". Thus, agents of heavier masses attract those of smaller mass. GSA uses exploration to navigate its space to guarantee choice of values by the agents are not violated, and uses exploitation to find optima in shortest time – with agents of heavier masses, moving slowly in order to attract those of lesser mass [28]. Agents are randomly initialized. At time *t*, a gravitational force of mass *j* acts on mass *i* based on R_{ij} Euclidean distance between any two masses as thus:

$$F_{ij}^{d} = G(t)\frac{M_i(t) * M_j(t)}{R_{ij}(t) + \varepsilon}\,(X_j^d(t) - X_i^d(t)) \quad -(1)$$

G (gravitation constant) decreases in time to control the search's accuracy with ε as the small constant. Thus, the total force is acting on each agent is given by:

$$F_i^d = \sum_{j \in kbest, j \neq 1} rand_j * F_{ij} \quad (2)$$

rand – randomizes agents' initial states at intervals [0,1] and acceleration of agent i, at time t in dimension d is directly proportional to force acting on that agent, and inversely proportional to its mass as:

$$A_i^d(t) = \frac{F_i^d(t)}{M_{ij}(t)} \quad -(3)$$

Next agent's velocity is a function of its current velocity and its current acceleration computed as:

$$V_i^d(t+1) = rand_i * V_i^d(t) + A_i^d(t) \quad -(4)$$

$$X_i^d(t+1) = X_i^d * V_i^d(t+1) \quad -(5)$$

$V_i^d(t)$ is agent velocity in *d*-dimension at time t, and rand is between [0,1]. Masses are calculated via fitness function and agents with heavier mass, attracts those of lesser mass, and the more efficient an agent in its solution. Masses are updated as:

$$M_i(t) = \frac{Fit_i(t) - worst(t)}{best(t) - worst(t)} \quad -(6)$$

$Fit_i(t)$ is fitness value of agent. Best(t) and worst(t) is strongest and weakest agent. For minimization task, best(t) and worst(t) are defined as:

$$best(t) = \min_{j \in \{1,2..N\}} Fit_i(t) \quad -(7)$$

$$worst(t) = \max_{j \in \{1,2..N\}} Fit_i(t) \quad -(8)$$

At start, agents are dispersed randomly in space. With each cycle, agents' positions, velocities and masses are updated via Eq. (4), (5) and (6). Algorithm stops if optimal is found or stop criterion is met. Via reverse engineering, agents of lower mass pull those of heavier masses for a minimization task.

4.2. HGSAGA as in Toroidal Queen

The initial use of GSA helps achieve a low fitness – so that once better individual is not found by GSA after a number of cycles, best individuals are chosen for GA operation via structured learning till an optimal solution is found. Factors defined for GSAGA includes (with GA), how many number of runs, how is population representation, its size and reproduction function – must be addressed [13].

GSA initializes algorithm with a 30-individual selected sub-pool. For GA, crossover and mutation is performed. Single point crossover is applied to mating individuals – in which, a number between 1 and 64 is randomly generated from Gaussian distribution, corresponding to the crossover point. It then

yields new individual (gene is combination of both parents), which undergoes mutation from where three random genes are selected for another mutation and are allocated new random values to replace the old pool with low fitness values (creating a new pool). This continues until an individual with a fitness value of zero (0) is found – to imply that the solution to the puzzle has been reached [3]. Mutation is applied, and model stops if best individual has fitness 0, solution found [10].

5. Hybrid Genetic Algorithm-Simulated Annealing (HGASA)

5.1. Simulated Annealing

SA as inspired by annealing, to strengthen glass and crystals – so that a glass is heated until it liquefies and allowed to slowly cool so that the molecules settles into lower energy states. Thus, it tracks and alters the state of an individual, constantly evaluating its energy via its energy function. Its optimal point is found by running series of Markov chain under different thermodynamic state. This *neighbouring* state is determined by randomly changing an individual's current state via a neighbourhood function. If a state with lower energy is found, individual moves to it; else, if neighbourhood state has a higher energy, individual moves to that state only, if an acceptance probability condition is met. If not met, individual remains at current state [9].

The acceptance probability is difference in energies between current and neighbouring states, and temperatures. Temperature is initially set high, so individual is more inclined towards higher energy state – allowing the individual to explore a greater portion of the space and preventing it from being trapped in local optimum. As model progresses – temperature reduces with cooling and individuals converge towards lowest energy states till an optimum point [9]. The algorithm is as thus:

Simulated Annealing Algorithm{}
1. Initialize individual state, energy and temperature
2. Loop until temperature is at minimum
3. Loop until maximum number of iterations reached
4. Find neighbourhood state via neighbourhood function
5. If neighbourhood state has lower energy than current
6. Then change current state to neighbouring state
7. Else if the acceptance probability is fulfilled
8. Then move to the neighbouring state
9. Else retain the current state
10. Keep track of state with lowest energy
11. End inner loop: End outer loop

5.2. Hybrid GASA

GASA is a powerful model that combines GA's parallel search to explore the space via multiple individuals and SA's flexibility in finding a better optimal point, even when a local minimum is found. Both are needed, to find solution [9, 11]. GA first yields low fitness – so that if a better individual is not found after number of runs, the best individual is chosen for a series of random walks until an optimal solution is found. Some factors must be defined: (1) On GA: number of runs, population representation, size and reproduction function, and (2) On SA (with GA complete), SA is run on the fittest individual until a solution is found and what is the neighbourhood size and function. As applied, a population of 64 is initialized. Only mutation function of GA is applied, to randomly select and randomly swap two unfixed cells (and if the GA produces an individual with a fitness close to the optimal – then temperature schedule is omitted and only one Markov chain is run). The number of mutations corresponds to the best fitness (best fitness and individual tracked until a fitness of 2 is found and experimentally, it is found that GA found a fitness of 2 very quickly).

[9] notes since GA found individuals with low energy, then enters SA cycle fairly late so that no temperature schedule is needed. Instead, a moderated Markov chain is used that accepts the states with energies of lower or equal to the current state's energy. This runs till state with energy of 0 is reached

(solution is found). SA and GA shares same fitness function; while the neighbourhood function in SA is same as mutation function used in GA.

6. Result Discussion

The result is displayed as in table 1 below as thus:

Table 1. Fitness Evaluation for Optimal Function

No of Queens	HPSOGA	CGA	HGSAGA	HGASA
8	235	288	202	391
20	5,964.5	5,669.7	6,024	2,043
50	25,789.1	14,991.4	11,227	9,879
100	99,032.8	103,439.9	98,208.2	44,578
200	195,643	195,657.6	240,991	96,747
500	243,786.3	306,799.4	423,765	132,987

HPSOGA found solution in 9seconds after 235 iterations. On a 20-queens task (to eradicate biasness), it found an optimal solution each time between 9seconds and 3minutes. Convergence time depends on fast velocities are updated, crossover and on the random mutation applied to individuals in the pool (Perez and Marwala, 2011).

CGA took 23seconds to find optimal after 288 moves. On each, it was able to find optimal solution – and time varied between 23seconds and 6minutes – and convergence time depends on how close the initial pool is to the solution and mutation applied.

GSAGA solved task in 5seconds after 202 moves. On each, GSAGA found optimal solution on a range between 5seconds and 3minutes – and convergence time depends on gravitational pull cum mass updates.

HGASA solves task at 2.112seconds after 391 moves. GA achieved a fitness of 2 in 90 iterations and SA used Markov chain of 301 iterations to find a solution. On each run, HGASA found optimal on a range between 2seconds and 3minutes – as its convergence time depends on initialization and the random swaps.

7. Conclusion and Recommendation

N-Queen problem is solved efficiently using stochastic techniques, four of which are used in this work. Solution space and fitness functions (common to all algorithms) with support/confidence model of ϖ1=0.2 and ϖ2=0.8 yields a better convergence as other suggested values, led to a slower or non-convergence.

8. References

[1] Abarghouei, A., Ghanizadeh, A and Shamsuddin, S., "*Advances in soft computing methods in edge detection"*, Journal of Advance Soft Computing Applications, vol. 1, no 2, 2009.

[2] Ahrens, W., "*Mathematische unterhaltungen und spiele"*, vol 1, B.G Tuebner, Leipzig, 1921.

[3] Cantu-Paz, E and Goldberg, D.E., "*Efficient parallel genetic algorithms: theory and practices"*, Computer in Applied Mechanics and Engineering, vol. 186, no 2, pp.221-238, 2000.

[4] Coello, C. A., Pulido, G. T and Lechuga, M.S., "*Multiple objectives with particle swarm optimization"*, Journal of Evolutionary Computation, Vol. 8, pp.256–279, 2002.

[5] Heppner, H and Grenander, U., "*Stochastic non-linear model for coordinated bird flocks"*, In Krasner, S (Ed.), "*The ubiquity of chaos",* pp.233–238, Washington: AAAS, 1990.

[6] Lewis, R., "*Metaheuristics can solve Sudoku",* Journal of Heuristics Archive, vol. 13, no. 8, pp.387 – 401, 2007.

[7] Ojugo, A.A., Eboka, A.O., Yoro, E.R., Okonta, E.O and Aghware, F.O., "*Genetic algorithm rule-based intrusion detection system"*, Journal of Emerging Trends in Computing and Information Systems, vol. 3, no.8, pp.1182-1194, 2012.

[8] Parsopoulos, K. E and Vrahatis, M., "*On computation of all global minimizers via particle swarm optimization"*, Transactions on Evolutionary Computation (IEEE), vol. 8, pp.211–224, 2004.

[9] Perez, M and Marwala, T., "*Stochastic optimization approaches for solving Sudoku",* Transaction on Evolutionary Computation (IEEE), pp.256–279, 2011.

[10] Reynolds, R., "*An introduction to cultural algorithms"*, Transaction on Evolutionary Programming (IEEE), pp.131-139, 1994.

[11] Kennedy, J., Eberhart, R. C and Shi, Y., "*Swarm intelligence"*. ISBN: 1-974-3564, San Francisco: Kaufmann, pp.35-87, 2001.

[12] Rivin, I., Vardi, I and Zimmermann, P., "*The n-queen problem"*, Electronic notes on mathematical and theoretical computer science, pp.629-639, 1994.

[13] Santos-Garcia, G and Palomino, M., "*Solving the Sudoku puzzle with rewriting rules"*, Electronic notes on Theoretical Computer Science, vol. 17, no.4, pp.79-93, 2007

[14] Clerc, M., "*The swarm and the queen: towards a deterministic and adaptive particle swarm optimization"*, In Proceedings of Evolutionary Computation (IEEE), Vol. 5, pp.123-132, 1999.

[15] Eberhart, R and Kennedy, J., "*New optimizer using particle swarm theory"*, In Proceedings of Symposium on Micro machine and human science (IEEE), Japan: Nagoya, 1995.

[16] Hassan, R and Crosswley, W., "*Variable population-based sampling for probabilistic design optimization and with a genetic algorithm"*, In Proceedings of the 42nd Aerospace Science meeting and conference (AIAA-2004-0452), Reno: NV, 2004.

[17] Hassan, R., Cohanin, B., De Wec and Venter, G., "*Comparison of PSO and GA",* In Proceedings of 44th Aerospace Science meeting (AIAA-2006-0562), Washington–DC, 2006

[18] Homaifar, A.A., Turner, J and Ali, S., "*N-queens problem and genetic algorithms"*, In Proceedings of IEEE Southeast conference (IEEE), pp.262-267, 1992.

[19] Hu, X., Eberhart, R.C and Kennedy, J., "*Solving constrained nonlinear optimization problems with particle swarm optimization",* In Proceedings of Multi-conference on Systems, Cybernetics and Informatics (IEEE), USA: Orlando, 2005a.

[20] Hu, X., Eberhart, R.C and Shi, Y., "*Swarm intelligence for permutation optimization: case study of n-queens"*, In Proceedings of Genetic and Evolutionary Computing Conference on Memetic algorithms (IEEE), pp.243 – 246, 2005b.

[21] Kennedy, J and Eberhart, R. C., "*Particle swarm optimization"*. In Proceedings on Neural networks (IEEE), pp.1942–1948, Honolulu, HI, Piscataway: IEEE, 1995.

[22] Kennedy, J and Eberhart, R. C., "*Discrete binary version of the particle swarm algorithm"*, In Proceedings on conference on Systems, Man and Cybernetics (IEEE), pp.4104–4109, 1997.

[23] Kennedy, J and Mendes, R., "*Population structure and particle swarm performance"*, In Proceedings of Congress on Evolutionary Computation (IEEE), pp.1671–1676, Honolulu: Piscataway, 2002.

[24] Kilic, A. and Kaya, M.A., "*New local search algorithm based on genetic algorithms for n-queens problem"*, In Proceedings on Genetics and Evolutionary Computation (IEEE), pp.158 – 161, 2001

[25] Mantere, T and Koljonen, J., "*Solving and rating Sudoku puzzles via genetic algorithm"*, In Proceedings on Evolutionary Computation (IEEE), pp.1382-1389, 2007.

[26] Poli, R., Wright A., McPhee, N and Langdon, W., "*Emergent behaviour, population-based search and low-pass filtering"*, In Proceedings of Congress on Computational Intelligence (IEEE), pp.395-402, Vancouver: Piscataway, 2006b.

[27] Dozier, G and Carlisle, A., "*Tracking changing extrema with particle swarm optimizer"*, Auburn University: Technical Report CSSE01-08, 2008.

[28] Ojugo, A.A., "*Gravitational search neural network algorithm for rainfall runoff modeling"*, Unpublished PhD thesis, Abakiliki: Ebonyi State University, 2009.

[29] History of Sudoku, Conceptis Editoria, [online]: www.conceptispuzzle.com/articles/sudoku, last date retrieved: Feb 13, 2013.

[30] Moraglio, A and Togelius, J., "*Geometric particles swarm optimization for Sudoku puzzle"*, 2007, [online] http://julian.togelius.com/Moraglio2007Geometric.pdf, last accessed 16-January-2013.

[31] Li, L.D. and Zhou, J., "Evolutionary Optimisation for Power Generation Unit Loading Application", *Progress in Intelligent Computing and Applications*, vol. 1, no. 1, pp. 37-49, 2012.