GENETIC ALGORITHM VERSUS PARTICLE SWARM OPTIMIZATION IN N-QUEEN PROBLEM

[*]Azhar W. Hammad, [**]Dr. Ban N. Thannoon

Al-Nahrain University/ College of Science/ Iraq

[*]azharalrawi@yahoo.com, [**]drban_2001@yahoo.com

Abstract

In recent year, particle Swarm Optimization (PSO) which is one of the area of evolutionary computations was developed, So in order to compare its performance, another popular optimization method Genetic Algorithm (GA) was chosen. The two methods employee different strategies and computation efforts.

In this paper, *N-Queens problem* was chosen to compare GA with PSO performance. GA with its own simple operators is stable in its performance under different search space sizes, while the PSO performs well in small search space size and its capabilities when space size becomes larger.

Keywords**:** Genetic Algorithm, Particle Swarm Optimization.

Introduction

In recent years, the area of Evolutionary Computation has come into its own. Two of the popular developed approaches are GAs and PSO, both of which are used in optimization problems. Although the two approaches are supported to find a solution to a given objective function, they employ different strategies and computational efforts. Therefore it is appropriate to compare their implementation.

The contribution in this work has two fields:

1. To compare the effectiveness of GA and PSO (i.e. compare their solution quality or convergence reliability).
2. To compare the efficiency of GA and PSO (i.e. compare their convergence speed).

The N-Queens problem is the problem of putting n queens on an n×n chessboard with ((n-k)!) search space such that non of them is to be able to attack any other. Eight and sixteen queens were tried in this implementation.

Genetic Algorithms

Genetic Algorithms are adaptive stochastic search algorithms (Stochastic searches are those that use probability to help guide their search) [1].

A GA is a powerful search technique that mimics natural selection and genetic operators. Its power comes from its ability to combine good pieces from different solutions and assemble them into a single super solution. GA can be distinguished from other search and optimization techniques by the fact that it is a process, which uses a population of many individuals, rather than a single one to solve a problem [2].

GAs are a family of computational models inspired by evolution. These algorithms encode a potential solution to a specific problem on a simple chromosome-like data structure and apply recombination operators to these structures so as to preserve critical information. They are often viewed as function optimizers, although the range of problems to which genetic algorithms have been applied is quite broad such as pattern recognition, image processing, machine learning, etc [3].

Elements of GAs

The GAs have the following elements and operators: Encoding, selection according to fitness, crossover to produce new offspring, and inversion [4].

i. Encoding

Suppose someone is seeking to find a solution to some problem. To apply a GA to that problem, the first thing he/she must do is to encode the problem as an artificial chromosome or chromosomes. These chromosomes can be string of 1s and 0s, parameter list, integer numbers, or even complex computer codes, but the key thing to keep in mind is that the genetic machinery will

manipulate a finite representation of the solution, not the solutions themselves [4].

ii. Selection

Another key element of GAs is the selection operator which is used to select chromosomes (called parents) for mating in order to generate new chromosomes (called offspring). In addition, the selection operator can be used to select elitist individuals. The selection process is usually biased toward fitter chromosomes. Selection methods are used as mechanisms to focus on apparently more profitable regions in the search space. One of these methods is tournament selection this selection way is more commonly used approach, a set of chromosomes (known as tournament size) are randomly chosen the fittest chromosome from the set is then placed in mating pool. This process is repeated until the mating pool contains a sufficient number of chromosomes to start the mating process.

iii. Crossover

Crossover is "the main explorative operator in GAs". Crossover occurs with a user-specified probability; called the crossover probability *Pc*. *Pc* is problem dependent with typical values in the range between 0.4 and 0.8.

One of the most popular type of crossover Partial Matched Crossover (PMX) is not suitable for binary coding problems. Under PMX, two strings aligned, and two crossing sites are picked uniformly at random along the strings. These two points define a *matching section* that is used to affect a cross through position-by-position exchange operations.

iv. Mutation

Mutation is performed after crossover by randomly choosing a chromosome in the new generation to mutate in order to explore new areas in the search space and to add diversity to the population of chromosomes to prevent being trapped in a local optimum.

Inversion is a different form of mutation. It is sometimes used in appropriate cases. Under inversion, two points are chosen along the length of the chromosome, the chromosome is cut at those points, and the end points of the cut switch places.

The GA Procedure

The following pseudo code of the GA illustrates the main steps that should be performed to produce the required solution.

```
Create an initial population of strings.
Calculate the fitness of each string.
While an acceptable solution is not found
    Select parent for next generation.
    Combine the parents to create new offspr
    Mutation and Inversion are applied
    according to some probability.
    Calculate the fitness of each offspring.
End While.
```

Pseudo code of Genetic Algorithm

Particle Swarm Optimization

Particle Swarm Optimization is one of the evolutionary computations, which can be used for optimization, developed by Kennedy and Eberhart in 1995.

This algorithm is based on the social behavior of individuals living together in groups such as bird flocking, fish schooling, and swarm of bees (or insects). A population of particles exists in the *n*-dimensional search space that the optimization problem lives in. Each particle has a certain amount of knowledge, and will move about the search space based on this knowledge. The particle has some inertia attributed to it and so will continue to have a component of motion in the direction it is moving. It also keeps track of the best solution for all the particles achieved so far, as well as the best solution achieved so far by each particle. The particle will then modify its direction such that it has additional components towards its own best position and towards the overall best position. This will provide some form of convergence to the search, while providing a degree of randomness to promote a wide coverage of the search space [5] [6].

PSO Topology

The common uses of PSOs are either *global* version or *local* version. In the global version of PSO, each particle flies through the search space with a velocity that is dynamically adjusted according to the particles of personal and the best performance achieved so far by all the particles. While in the local version of PSO, each particle's velocity is adjusted according to its personal best and the best performance achieved so far within its neighborhood. The neighborhood of each particle is generally defined as topologically nearest particle to the particle at each side [7].

A lot of researches had work on improving PSO performance by designing or implementing different types of neighborhood structures. Each neighborhood structure has its strength and weakness. It works better in one kind of problems, but worse on the other kind of problems. When using PSO to solve a problem, not only the problem needs to be specified, but the neighborhood structure of the PSO utilized, should also be clearly specified [7].

PSO Algorithm

As described by Kennedy and Eberhart, the PSO algorithm is an adaptive algorithm based on a social-psychological metaphor; a population of individuals (referred to as particles) adapts by returning stochastically toward previously successful regions.

Particle Swarm has two primary operators: Velocity update and Position update. During each iteration, each particle is accelerated toward the particle's previous best position and the global best position. At each iteration a new velocity value for each particle is calculated based on its current velocity, the distance from its previous best position, and the distance from the global best position. The new velocity value is then used to calculate the next position of the particle in the search space. This process is then iterated a set number of times or until a minimum error is achieved [5] [8].

The PSO algorithm depends on its implementation in the following two relations [7]:

The velocity of particle *i* is updated using the following equation [8]:

$$v_{id}(t+1) = wv_{id}(t) + c_1r_1(t)(p_{id}(t) - x_{id}(t)) + c_2r_2(t)(p_{gd}(t) - x_{id}(t)) \quad \text{.....(1)}$$

The position of particle *i*, x_i is then updated using the following equation:

$$x_{id}(t+1) = x_{id}(t) + v_{id}(t+1) \quad \text{.....(2)}$$

where:

t is the current time step, t+1 is the next time step.

x_{id} (t) is the current state (position) at site d of individual *i*.

v_{id} (t) is the current velocity at site *d* of individual *i*.

w(t) is the inertia weight

$$w = ((T_{max} - G) * (0.9 - 0.4) / T_{max}) + 0.4$$

Where $\boldsymbol{T_{max}}$ is the maximum number of Iterations and $\boldsymbol{G}$ is the current generation number.

p_{id} is the individual's i best state (position) found so far at site d.

p_{gd} is the neighborhood best state found far at site d.

c_1 cognition parameter 1, a positive constant, usually set to 2.0.

c_2 social parameter 2, a positive constant, usually set to 2.0.

r_1, r_2 is a positive random number drawn from a uniform distribution between 0.0 and 1.0.

The pseudo code of the PSO

The following pseudo code illustrates the main steps of the PSO.

```
For each particle
    Initialize particle
End
Repeat
    For each particle
        Calculate fitness value
        If the fitness value is better than
        best fitness value (Pid) in history
        Set current value as the new Pid
    End
    Choose the particle with the best
      fitness value of all the particles as
      the Pgd
    For each particle
        Update particle velocity
          according to equation (1)
        Update particle position
          according to equation (2)
    End
        Until Stopping criteria
```

Pseudo code of PSO

N-Queens Problem

Problem Definition: The N-Queens problem is analogue of the classic 8-Queens problem from chess. There is an n×n chessboard and the goal is to have N Queens placed into board squares so that no queen is attacking the other so its search space size=((n-k)!) where k=1..n-1.

Developing a Coding Scheme

This problem can be generalized as placing *n* nonattacking queens on an *n*×*n* chessboard. Since each queen must be on a different raw and column, we can assume that queen *i* is placed in i-th column. All solutions to the *N*-queens problem can therefore be represented as string of *n* bits (i.e String length = n).

i. Initialization

The initial population in each GA and PSO algorithm is seeded by randomly generated individuals. In our works the maximum number of generation is dependent on the problem itself. The population/swarm size (i.e. individuals/particles numbers) is set to 10, 20, 30, and 100. Ten runs were performed for each population/swarm size, and the population/swarm size not changing during the evolutionary search

Creating a Fitness Function

The position of a number in the string represents queen's column position, while its value represents queen's row position, so the row and column conflicts are already satisfied. A fitness function can be designed to count diagonal conflicts: more conflicts there are, worse the solution. For correct solution, the function will return Zero.

A queen that occupies *i*-th column and q_i-th row is located on ($i+q_i-1$) left and ($n-i+q_i$) right diagonal. A fitness function first allocates counters for all diagonal then, for each queen, counters for one left and one right diagonal that queen occupies are increased by one. After evaluation, if a counter has a value greater than 1, there are conflicts on the corresponding diagonal.

Results

Tables (1) and (2) describe the GA and PSO operators and parameters that are used in solving N-queens problem.

Table (1)
GA's operators and parameters for solving N-Queens Problem

Initialization	Random
Representation	Integer String of Length= n
Selection	Tournament Selection
Recombination	Partial Matched Crossover (PMX)
Mutation	Swap
Mutation Probability	0.5
Population Size	10, 20, 30,…, 100
Maximum Number of generations (NoG)	100
Stopping Condition	Solution or Number of Generations= NoG

Table (2)
Describes PSO parameters for solving N-Queens Problem

Initialization	Random
Representation	Integer String of Length= n
w	w= ((T_{max} - G) * (0.9 - 0.4) / T_{max}) + 0.4
c1 and *c2*	2.0
r1 and r2	Random [0..1]
Swarm Size or Number of Particles	10, 20, 30,…, 100
Maximum Number of Iterations(T_{max})	100, 30000
Stopping Condition	Solution or Number of iterations= T_{max}

For ten times running and for 10, 20, 30, …, 100 Population/Swarm size, the average of the best three results for 8-queens are presented, Table (3) illustrates that PSO give us best solutions than GA in number of generations and time.

Table (3)
The average of comparison results between GA and PSO to solve 8-Queens.

Pop Size	GA		PSO	
	No. of Genera-tions	Time/Sec	No. of Itera-tions	Time/Sec
10	12	1.8485	13	1.1977
20	7	1.7053	4	1.3741
30	3	1.5430	2	1.5269
40	3	1.3966	4	1.4405
50	5	1.4659	3	1.3814
60	2	1.4373	3	1.4725
70	2	1.9066	4	1.6210
80	3	1.9255	3	1.8541
90	3	2.3777	4	2.2561
100	3	2.4925	3	2.1666

But, for 16-queens, table (4) shows that solutions found using GA are better than those found using PSO.

Table (4)
The average of comparison results between GA and PSO to solve 16-Queens.

Pop Size	GA		PSO	
	No. of Genera-tions	Time/ Sec	No. of Itera-tions	Time/Sec
10	60	1.9999	23547	61.9999
20	70	2.9999	18444	95.9999
30	39	2.3333	10003	76.9999
40	31	3	143366	147.6666
50	25	2.3333	15150	194.6665
60	24	2.6666	13247	203.9999
70	35	4.6666	3853	69.6666
80	36	4.9999	4443	91.3333
90	30	4.6666	7160	167.3333
100	39	6.6666	4548	117.6666

Discussion

The PSO Algorithm shares similar characteristic to GA, however manner in which the two algorithms traverse the search space is fundamentally different.

Both Genetic Algorithms and Particle Swarm Optimizers share common elements:

1. Both initialize a population in random manner.
2. Both use an evaluation function to determine how fit (good) a potential solution is.
3. Both are reproduction of the population based on fitness values.
4. Both are generational, that is both repeat the same set of processes for a predetermined amount of time.
5. Both are stopping when requirements are met.

However, PSO does not have genetic operators like crossover and mutation. Particles update themselves with internal velocity. They also have memory, which is important to the algorithm. The information sharing mechanism in PSO is significantly different. In GAs, chromosomes share information with each other. So the whole population moves like a one group towards an optimal area. In PSO, only best neighborhood gives out the information to others. It is a one way information sharing mechanism. The evolution only looks for the best solution. All the particles tend to converge to the best solution quickly even in the local version in most cases.

Conclusion

For Non-linear problems (Solving N-Queen problem with n=8 and n=16) with ((8-k)! and (16-k)!), search space size respectively, we conclude that GA with its own simple operators (selection, crossover and mutation), was stable in its performance under different search space sizes, the GA can reaches optimal or near optimal solution.

While the PSO, performs well in small search space size but decreases its capabilities with more complicated problems, i.e., when it has large search space size.

References

[1] K.Grant, "An Introduction to Genetic Algorithms", C++ user Journal, March 1995.

[2] M.Baright, J.Timmins, G.Heliker, "Multiobjective Optimization of Control Systems Via Genetic Algorithms", University of Illinois, IlliGAL Report NO. 07009, 1997.

[3] D.Whitley," A. Genetic Algorithms Tutorial", Colorado State University, Fort Collins, Co 80523, 1994, whitley@cs.colostate.edu.

[4] D. E. Goldberg, Genetic Algorithms in search, optimization, and Machine Learning, Addison-Wesley Publishing company, INC., 1989.

[5] J. Kennedy and R. Eberhart, Swarm Intelligence. Morgan Kaufmann Publishers, Inc., San Francisco, CA, 2001.

[6] C.R. Mouser, S.A. Dunn, "Comparing Genetic Algorithms and Particle Swarm Optimization for Inverse Problem Exercise", Austral. Mathematical Soc, 2005.

[7] Y. Shi, "Particle Swarm Optimization", Electronic Data Systems, Inc. Kokomo, IN 46902, USA Feature Article, IEEE Neural Networks Society, February 2004.

[8] M. Settles, "An Introduction to Particle Swarm Optimization", Department of Computer Science, University of Idaho, Moscow, Idaho U.S.A. 838, November 7, 2005.