Swarm Intelligence for Permutation Optimization: A Case Study of n-Queens Problem

Xiaohui Hu [1, 2] Russell C. Eberhart [2] Yuhui Shi [3]

[1] Department of Biomedical Engineering
Purdue University, West Lafayette, Indiana, USA
hux@ecn.purdue.edu

[2] Department of Electrical and Computer Engineering
Purdue School of Engineering and Technology, Indianapolis, Indiana, USA
reberhar@iupui.edu

[3] EDS Embeded Systems Group
Kokomo, Indiana, USA
Yuhui.Shi@eds.com

Abstract- This paper introduces a modified Particle Swarm Optimizer which deals with permutation problems. Particles are defined as permutations of a group of unique values. Velocity updates are redefined based on the similarity of two particles. Particles change their permutations with a random rate defined by their velocities. A mutation factor is introduced to prevent the current pBest from becoming stuck at local minima. Preliminary study on the n-queens problem shows that the modified PSO is promising in solving constraint satisfication problems.

I. INTRODUCTION

A permutation problem is a constraint satisfaction problem with the same number of variables as values, in which each variable takes a unique value. Any solution can be thought of as assigning a permutation to the variables. When a permutation satisfies all the constraints, it is considered a feasible solution. For a permutation problem, there might be one or multiple feasible solutions. The n-queens problem is one of the best examples of permutation problems. Permutation optimization problems have been found in many areas. There are many techniques developed to handle permutation problems. In this paper, a new method called particle swarm optimization (PSO) is introduced to handle the permutation problems.

The n-queens problem consists of placing n queens on an N by N chess board, so that they do not attack each other, i.e. on every row, column or diagonal, there is only one queen exists. It is a classical complex constraint satisfaction problem in the artificial intelligence (AI) area. It has been used as a benchmark for developing new AI search techniques. During the last three decades, the problem has served as an example and benchmark for backtracking algorithms, permutation generation, the divide and conquer paradigm, constraint satisfaction problems, neural networks, and genetic algorithms. Also, the n-queens problem has many practical applications such as VLSI testing, air traffic control, modern communication systems, data/message routing, load balancing in multiprocessor computers, data compression, computer task scheduling, and optical parallel processing [1]. The n-queens problem has three variants: finding one solution, finding a family of solutions, and finding all solutions. This paper deals with finding one solution within a family.

PSO is a population based stochastic optimization technique developed by Eberhart and Kennedy in 1995, inspired by social behavior of bird flocking or fish schooling [2, 3]. During the past several years, PSO has been successfully applied to multidimensional optimization problems [4], artificial neural network training [5-7], and multiobjective optimization problems [8-10]. However, there is no research on permutation optimization reported in the literature.

The rest of the paper is organized as follows: Section II reviews the basic forms of particle swarms. Section III describes the new methods for velocity update and particle update to handle the permutation parameter set. Section IV describes the n-queens problem, and Section V summarizes the experimental results.

II. PARTICLE SWARM OPTIMIZATION

Similar to Genetic Algorithms (GAs), PSO is a population based optimization tool. The system is initialized with a population of random solutions and searches for optima by updating potential solutions over generations. However, unlike GA, PSO has no evolution operators such as crossover and mutation. In PSO, the potential solutions, called particles, "fly" through the problem space by following the current better-performing particles.

Each particle keeps track of its coordinates in the problem space which are associated with the best solution (fitness) it has achieved so far. (The fitness value is also stored.) This value is called *pbest*. Another "best" value that is tracked by the particle swarm optimizer is the best value, obtained so far by any particle in the neighborhood of the particle. This location is called *nbest*. When a particle takes all the population as its topological neighbors, the best value is a global best and is called *gbest*.

```
Initialize the population
Do {
    For each particle {
        Calculate fitness value
        If the fitness value is better than the best fitness
        value (pBest) in history
            Set current value as the new pBest
    }
    Choose the particle with the best fitness value of all the
    topological neighbor particles as the nBest
    For each particle {
        Calculate new velocity
            $\vec{V}_{new} = w \times \vec{V}_{old} + c_1 \times rand() \times (\vec{P}_{pBest} - \vec{X}) +$
            $c_2 \times Rand() \times (\vec{P}_{nBest} - \vec{X})$
        Update particle position
            $\vec{X}_{new} = \vec{X}_{old} + \vec{V}_{new}$
    }
} Until termination criterion is met
```

Figure 1: Procedure of PSO

The particle swarm optimization concept consists of, at each time step, changing the velocity of (accelerating) each particle toward its *pbest* and *nbest* locations (local version pf PSO). Acceleration is weighted by a random term, with separate random numbers being generated for acceleration toward *pbest* and *nbest* locations. Figure 1 shows the typical procedure of PSO.

One of the reasons that particle swarm optimization is attractive is that there are few parameters to adjust. One version, with slight variations, works well for a wide variety of applications. Particle swarm optimization has been used for approaches that can be used across a wide range of applications, as well as for specific applications focused on a specific requirement.

III. DEALING WITH PERMUTATION SET

In traditional PSO, each particle represents a solution in the parameter space. The particle is encoded as a string of positions, which represent a multidimensional space. All the dimensions typically are independent of each other, thus the updates of the velocity and the particle are performed independently in each dimension. This is one of merits of PSO. However, it is not applicable for permutation problems since the elements are not independent of each other. It is possible that two or more positions can get the same value after the update, which breaks the permutation rule. Thus the conflicts must be eliminated. Here a new particle update strategy is proposed.

In traditional PSO, the velocity is added to the particle on each dimension to update the particle, thus it is a distance measure. If the velocity is larger, the particle may explore more distant areas. Similarly, the new velocity in the permutation scenario represents the possibility that the particle changes. If the velocity is larger, the particle is more likely to change to a new permutation sequence. The velocity update formula remains the same. However the velocity is limited to absolute values since it only represents the difference between particles. The particle update process is changed as follows: the velocity is normalized to the range of 0 to 1 by dividing it by the maximum range of the particle. Then each position randomly determines if there is a swap with a probability determined by the velocity. If a swap is required, the position will set to the value of same position in *nBest* by swapping values. This process is shown in Figure 2.

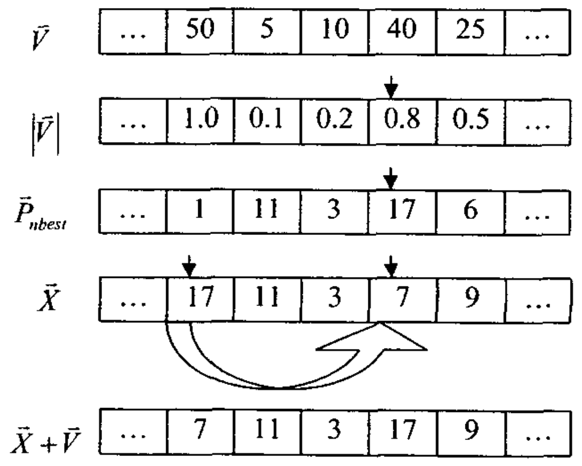

Figure 2: Particle update

Mutation is introduced due to the shortcoming of the above modification of PSO. Since the particle tries to follow the same sequence as *nBest*, it would stay in its current position forever when it was identical to *nBest*. So a new kind of mutation factor is introduced. The particle will randomly swap one pair of positions in the permutation as shown in Figure 3 if it is identical to *nBest*.

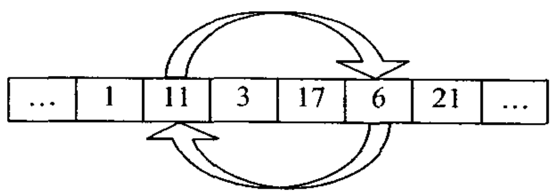

Figure 3: Particle mutation

In the following section, the n-queens problem is used to test the performance and validity of the new velocity and particle update technique.

IV. N-QUEENS PROBLEM

In this study, n-dimension permutations are used to represent the solution of the n-queens problem. Each particle uses a permutation of n numbers from 1 to N as the potential solution. The *i*th number of the permutation represents the column position in the *i*th row of the chessboard. To illustrate how it appears in the population, the particle for N=6 problems may be the following: 3 6 2 4 1 5. The first number means the first queen is at the third position in the first row, the second number means the second queen is at the sixth position in the second row, and so on. Figure 4 shows a translation from the permutation to the chessboard positions.

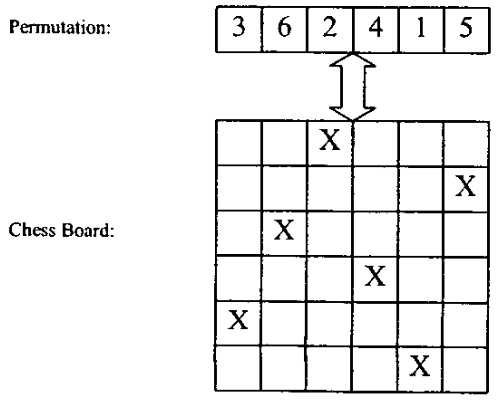

Figure 4: Permutation representation of n-queens problem

By using permutations, the horizontal and vertical conflicts of the queens are eliminated [11]. Thus to find a solution, the objective is to eliminate the diagonal conflicts. The fitness function is defined as the number of conflicts or collisions along the diagonals of the board. The objective is changed to minimize the number of conflicts or collisions. The fitness value of an ideal final solution should be zero.

V. EXPERIMENTAL RESULTS

In PSO, the parameters were set as follows: the population size was 10, the local version of PSO was used and the neighborhood size was 2. The maximum velocity was set to the range of the permutation. The inertia weight was [0.5 + (Rnd/2.0)]. The learning rates were 1.49445.

Figure 5 shows the results for the problems of 10 to 200 queens. Each parameter combination was run 100 times and the results represent the mean number of function evaluations to reach a solution. From the results, it can be seen that PSO successfully finds a solution of the n-queens problems in a short time. Furthermore, the numbers of function evaluations increase near linearly as the numbers of queens increase. Table 1 shows some comparisons and it can be seen that this method is competitive with GA based algorithms [1, 11].

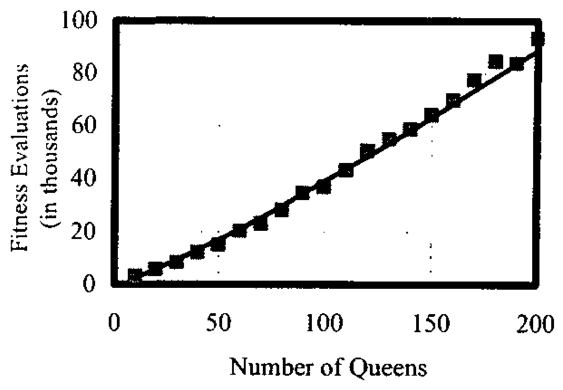

Figure 5: Number of fitness evaluations needed for different number of n-queens problem

Table 1: Comparison of the results for different n-queens approaches.

Number of Queens	Fitness evaluations needed to find a solution		
	This paper	Homaifar [11]	Kilic* [1]
20	5,669.7	2,043	6,024
50	14,991.4	59,227	19,879
100	36,799.4	244,208	44,578
200	93,439.9	340,991	86,747

* the numbers used here are digitized from the basic GA results of the Figure illustrated in [1]. They are not accurate.

VI. CONCLUSIONS

The purpose of the study was to determine how well PSO handles permutation parameter sets. The n-queens problem was used to test the performance and validity of the new technique. The performance of PSO compares well with genetic algorithms. It demonstrated that PSO is effective to handle n-queens problem. However, it still needs to be verified whether this approach can be extended to other combinatorial or constraint satisfaction problems.

References

[1] Kilic, A. and Kaya, M. A new local search algorithm based on genetic algorithms for the n-queen problem. Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2001) Second Workshop on Memetic Algorithms (2nd WOMA), pp. 158-161, 2001.

[2] Eberhart, R. C. and Kennedy, J. A new optimizer using particle swarm theory. Proceedings of the Sixth International Symposium on Micromachine and Human Science, Nagoya, Japan. pp. 39-43, 1995.

[3] Kennedy, J. and Eberhart, R. C. Particle swarm optimization. Proceedings of IEEE International Conference on Neural Networks, Piscataway, NJ. pp. 1942-1948, 1995.

[4] Shi, Y. and Eberhart, R. C. A modified particle swarm optimizer. Proceedings of the IEEE Congress on Evolutionary Computation (CEC 1998), Piscataway, NJ. pp. 69-73, 1998.

[5] Eberhart, R. C. and Shi, Y. Evolving artificial neural networks. Proceedings of International Conference on Neural Networks and Brain, 1998, Beijing, P. R. China. pp. PL5-PL13, 1998.

[6] Eberhart, R. C. and Hu, X. Human tremor analysis using particle swarm optimization. Proceedings of the IEEE Congress on evolutionary computation (CEC 1999), Washington D.C. pp. 1927-1930, 1999.

[7] van den Bergh, F. and Engelbrecht, A. P. Training product unit networks using cooperative particle swarm optimisers. Proceedings of INNS-IEEE International Joint Conference on Neural Networks 2001, Washington DC, USA. 2001.

[8] Coello Coello, C. A. and Lechuga, M. S. MOPSO: a proposal for multiple objective particle swarm optimization. Proceedings of the IEEE Congress on Evolutionary Computation (CEC 2002), Honolulu, Hawaii USA. 2002.

[9] Hu, X. and Eberhart, R. C. Multiobjective optimization using dynamic neighborhood particle swarm optimization. Proceedings of the IEEE Congress on Evolutionary Computation (CEC 2002), Honolulu, Hawaii USA. 2002.

[10] Parsopoulos, K. E. and Vrahatis, M. N. Particle swarm optimization method in multiobjective problems. Proceedings of the ACM Symposium on Applied Computing 2002 (SAC 2002), pp. 603-607, 2002.

[11] Homaifar, A. A., Turner, J., and Ali, S. The n-queens problem and genetic algorithms. Proceedings of the IEEE Southeast Conference, pp. 262-267, 1992.