2000

Multi-phase discrete particle swarm optimization

Buthainah Sabeeh No'man Al-kazemi
Syracuse University, balkazem@syr.edu

Chilukuri K. Mohan
Syracuse University, ckmohan@syr.edu

Multi-phase Discrete Particle Swarm Optimization

Buthainah Al-kazemi and Chilukuri K. Mohan
2-177 CST, Dept. of Electrical Engineering and Computer Science
Syracuse University, Syracuse, NY 13244-4100
balkazem/ckmohan@syr.edu

Abstract

This paper describes a successful adaptation of the Particle Swarm Optimization algorithm to discrete optimization problems. In the proposed algorithm, particles cycle through multiple phases with differing goals. We also exploit hill climbing. On benchmark problems, this algorithm outperforms a genetic algorithm and a previous discrete PSO formulation.

1. Introduction

The *Particle Swarm Optimization (PSO)* algorithm has been successful in solving a number of continuous optimization problems [1]. Unlike genetic algorithms and evolutionary programming, each individual (*particle*) in PSO traces a trajectory in the search space, constantly updating a *velocity* vector based on the best solutions found so far by that particle as well as others in the population (*swarm*) [2, 3].

The PSO algorithm was originally proposed for continuous problems, and attempts have been made recently to extend it to discrete optimization problems. Kennedy and Eberhart [4] proposed the first discrete version (referred to as *Quantum DiPSO* in [5]) and Clerc [6] has shown promising results on variants of the PSO specialized for some constrained optimization problems such as TSP. Our preliminary experiments with various Discrete PSO variants are reported in [5].

This paper formulates the *Multi-phase Discrete PSO (M-DiPSO)* algorithm, which incorporates hill climbing using random-sized steps in the search space. The particles in the swarm are divided into groups that follow different search strategies.

In section 2, we explain the original PSO algorithm and the Quantum DiPSO algorithm [1]. Section 3 describes the M-DiPSO algorithm. Section 4 contains details of the experiments, and Section 5 describes the results we obtained on benchmark problems.

2. Particle Swarm Optimization

Each swarm contains *M* particles with randomly initialized positions and velocities in an *n*-dimensional search space. Each particle stores its current position *(x)*, current velocity *(v),* and the best position it has visited so far (*l),* and (in the most frequently used version) is also presumed to know the best *(g)* among all positions visited so far by all particles in the swarm. The particle moves according to the dynamics described below.

1- The velocity is incremented by a weighted sum of *(l-x)* and *(g-x);* the magnitude of each velocity component is often limited (to 5 in the experiments reported in this paper).
2- The position is incremented by the velocity.

Particles move until termination criteria are met, such as a user-defined maximum number of iterations (discrete time steps) or a satisfactory solution is found [1].

For binary discrete search spaces, Kennedy and Eberhart [4] use the above velocity update equation but compute the actual new position component to be 1 with a probability obtained by applying a sigmoid transformation *(1/(1+exp(-v))* to the velocity component. We refer to this as the Quantum DiPSO or Q-DiPSO algorithm [5].

3. Multiphase DiPSO (M-DiPSO)

The M-DiPSO algorithm adopts the following equation for updating the *i*th component of the velocity of particle *m:*

$$v_{m,i}(t+1) = C_v * v_{m,i}(t) * + C_x * x_{m,i}(t) + C_g * g_i(t)$$

where the signs of the coefficients determine the direction of the particle movement. At any given time, each particle is in one of the possible *phases,* determined by its preceding phase and the number of iterations executed so far. Within each phase, particles fall into different groups with different coefficient values for each group.

The M-DiPSO algorithm is fairly robust and gives similar results for many different parameter values; the results shown here are with the smallest non-trivial number of groups and phases, two for each.

In our experiments, the respective coefficients were chosen to be as follows. In phase 1, each particle in the first group uses coefficients (1, -1, 1), moving toward *g*, the global best position found so far by all the particles, whereas velocities of particles in the second group are changed in the opposite direction, using coefficients (1, 1, -1). In phase 2, the coefficients for the two groups are switched, i.e., (1, 1, -1) for the first group and (1, -1, 1) for the second group.

These coefficients have been successful in solving the benchmark problems on which experiments have been carried out, although other coefficient values are also possible. Variants of the algorithm may invoke more groups and phases.

Switching from phase to phase can be done using one of two methods. The first method is controlled by a parameter *p*, so that *N/p* of the iterations are in each phase. The second method, used for the results reported in this paper, is adaptive and performs better: phase change occurs if no global best fitness improvement is observed in *S* recent steps of the current phase. In experiments reported here, parameter *S* was chosen to be 5.

Periodically, the velocity vectors are randomly reinitialized at a user-determined frequency. For fair comparison, experiments with the GA as well as Q-DiPSO invoked periodic random re-initialization, but this had little effect on the results reported below.

Unlike previous PSO variants, M-DiPSO incorporates hill-climbing by permitting particle position to change only if such a change improves fitness. Each particle's current position is better than its previous positions, hence the update equations do not contain a separate term corresponding to the local best (*l*), thus differing from the original PSO update equations. Hill climbing tests one dimension at a time which requires a large number of fitness evaluations. Instead the M-DiPSO algorithm successively updates a randomly chosen fraction of consecutive velocity vector components, and then updates the corresponding position vector components after testing that such an update improves fitness (or quality measure being optimized). The first *s* components are first updated, then the next *s*, and so on. In our experiments, *s* was chosen randomly from the interval [1,10]. If the current position cannot be improved by flipping any subset of *s* consecutive bits, the particle can remain stuck in its current position, failing to find a global optimum; to overcome this potential problem arising from high epistasis, the range of values from which *s* is chosen may be steadily increased with the number of iterations

4. Experiments

A traditional GA (using one-point crossover with crossover probability 0.9 and mutation rate the reciprocal of string-length), the Q-DiPSO algorithm [1], and MDiPSO were tested on three different benchmark problems, described below (see [7] for more detailed descriptions). These are all deceptive problems with a large number of local optima. In the following, $|x|$ denotes the sum of the bits in a substring *x*.

1. Goldberg's order-3 deceptive problem: The fitness of a bit-string is the sum of the result of separately applying the following function to consecutive groups of three components each:

$$f_3(x) = \begin{cases} 0.9 & if \quad |x| = 0 \\ 0.6 & if \quad |x| = 1 \\ 0.3 & if \quad |x| = 2 \\ 1.0 & if \quad |x| = 3 \end{cases}$$

2. Bipolar order-3 problem: The fitness is the sum of the result of applying the following function to consecutive groups of six components each:

$$f_6(x) = \begin{cases} 1.0 & if \quad |x| = 0 \quad or \quad 6 \\ 0.0 & if \quad |x| = 1 \quad or \quad 5 \\ 0.4 & if \quad |x| = 2 \quad or \quad 4 \\ 0.8 & if \quad |x| = 3 \end{cases}$$

3. Mühlenbein's order-5 problem: The fitness is the sum of the results of applying the following function to consecutive groups of five components each:

$$f_5(x) = \begin{cases} 4.0 & if \quad x = 00000 \\ 3.0 & if \quad x = 00001 \\ 2.0 & if \quad x = 00011 \\ 1.0 & if \quad x = 00111 \\ 3.5 & if \quad x = 11111 \\ 0 & otherwise \end{cases}$$

5. Results

Figure 1 and Figure 2 show how M-DiPSO scales up with problem size, using 1000 iterations, 30 particles and increasing the number of dimensions from 30 to 300 for Goldberg's order-3 deceptive problem. In the

table, we list problem dimensionality *(n)*, best fitness reached and the fitness count (number of fitness evaluations, averaged over 10 trials) needed to reach the best fitness. In each case, M-DiPSO reaches the best fitness in each trial using a small number of fitness evaluations that appears to increase sub-quadratically.

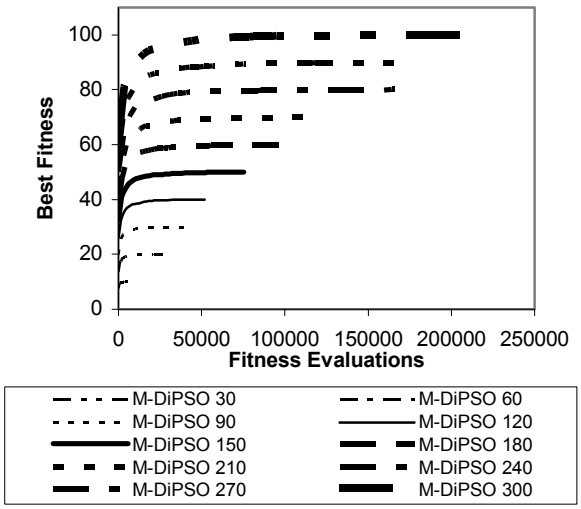

Figure 1: Evolution of best fitness plotted against the number of fitness evaluations for M-DiPSO algorithm applied to Goldberg's order-3 deceptive problems of different dimensionality.

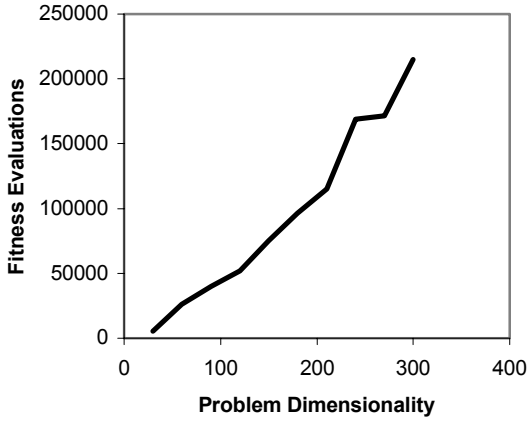

Figure 2: Performance of M-DiPSO for Goldberg's order-3 problems ranging in dimensionality from 30 to 300; results averaged over 10 trials

Tables 1, 2 and 3 show the results for the three different problems, respectively. In each table, we list the algorithm used, problem dimension *(n)*, and the number of fitness evaluations required to reach the best fitness listed. All results were averages over 10 trials. In every problem, M-DiPSO reached the best fitness requiring the fewest fitness evaluations. The GA and Q-DiPSO were executed for 100,000 fitness evaluations in each experiment, and terminated without reaching the optimal fitness value. The numbers of fitness evaluations required by M-DiPSO to find the optimal solutions are shown in parentheses.

Figures 3, 4 and 5 compare the performance of the three algorithms. M-DiPSO uses population size of 30 and the rest of algorithms use population size of 100. All the algorithms were terminated after 100,000 fitness evaluations. These results were averages over 10 trials. As clearly apparent from the figure, M-DiPSO reached the best fitness requiring far fewer fitness evaluations than the other algorithms for all the problems, using a smaller population size.

On a 60-bit non-deceptive variant of Goldberg's order-3 problem, with f_3 (001)= f_3 (010)= f_3 (100)=0.3 and f_3 (011)= f_3 (101)= f_3 (110)=0.6, M-DiPSO, GA and Q-DiPSO required 5978, 43326 and 52800 fitness evaluations, respectively, to reach the global optimum, showing that deception is not necessary for M-DiPSO to outperform the GA.

Problem Dimensionality	**30**	**60**	**90**	**150**
GA	9.42	24.05	24.05	36.74
Q-DiPSO	9.44	16.8	24.02	37.13
M-DiPSO (evals)	10 (5417)	20 (26368)	30 (39885)	50 (75150)

Table 1: Best fitness found using GA, Q-DiPSO and M-DiPSO for Goldberg's order-3 deceptive problem instances of different dimensionality, averaged over 10 trials.

Problem Dimensionality	**30**	**60**	**90**
GA	4.68	8.48	11.92
Q-DiPSO	4.66	8.36	11.72
M-DiPSO (evals)	5 (15690)	10 (45902)	15 (83348)

Table 2: Best fitness found using GA, Q-DiPSO and M-DiPSO for Bipolar problem instances of different dimensionality, averaged over 10 trials.

Problem Dimensionality	**30**	**60**	**90**	**150**
GA	23.05	36.3	47.9	62.25
Q-DiPSO	21.65	34.8	44.5	64.2
M-DiPSO (evals)	24 (5354)	48 (15344)	72 (42358)	120 (88488)

Table 3: Best fitness found using GA, Q-DiPSO and M-DiPSO for Mühlenbein's problem instances of different dimensionality, averaged over 10 trials.

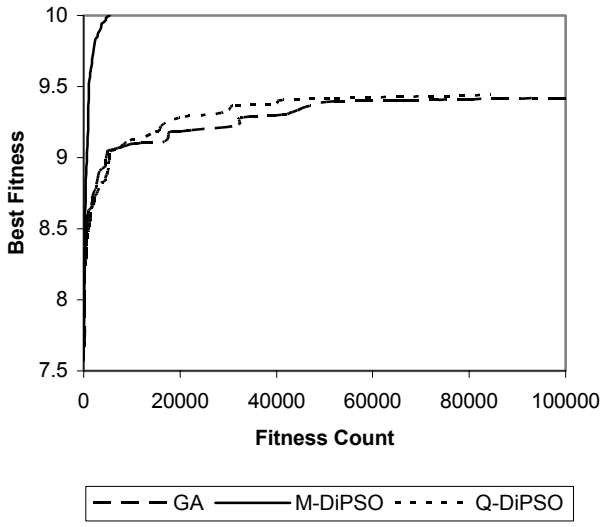

Figure 3: Comparison of GA, Q-DiPSO and M-DiPSO for Goldberg's 30-bit problem

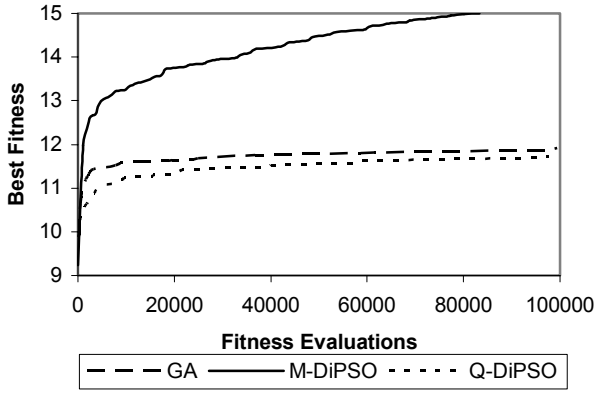

Figure 4: Comparison of GA, Q-DiPSO and M-DiPSO for BiPolar 90-bit problem

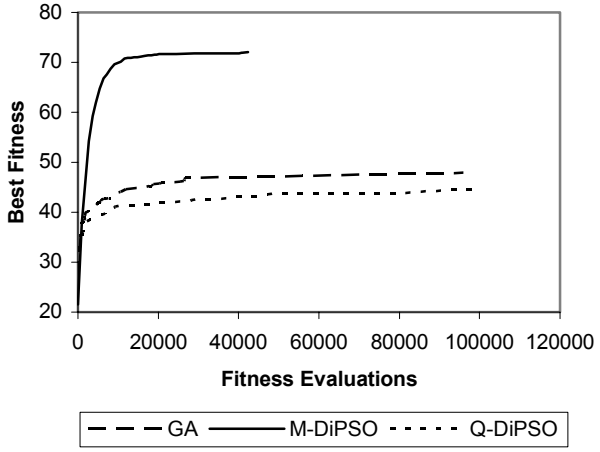

Figure 5: Comparison of GA, Q-DiPSO and M-DiPSO for Mühlenbein's 90-bit problem

6. Concluding Remarks

The results presented in the previous section show that M-DiPSO is a powerful algorithm that succeeds in solving the benchmark binary optimization problems using a small number of fitness evaluations to reach the best fitness. It also requires only a small number of particles. Although not shown in the tables, the actual computation times required were also small compared to other algorithms.

Current work involves more exhaustive testing of M-DiPSO for other benchmark problems. We are also exploring its application to continuous space problems. Benchmark continuous space problems can also be solved using the discrete version of the algorithm, encoding real numbers using bit strings. Preliminary results have shown that M-DiPSO can succeed in solving such problems

References

[1] J. Kennedy and R. C. Eberhart, "Particle Swarm Optimization," Proc. IEEE International Conference on Neural Networks, 1995.

[2] E. Ozcan and C. K. Mohan, "Particle swarm optimization: surfing the waves," Proc. Congress on Evolutionary Computation, Piscataway, NJ, 1999.

[3] Y. Shi and R. Eberhart, "A Modified Particle Swarm Optimizer," Proc. IEEE International conference on Evolutionary Computation, Anchorage, Alaska, 1998.

[4] J. Kennedy and R. C. Eberhart, "A discrete binary version of the particle swarm algorithm," Proc. Conf. on Systems, Man, and Cybernetics, Piscataway, NJ, 1997.

[5] C. K. Mohan and B. Al-kazemi, "Discrete particle swarm optimization," Proc. Workshop on Particle Swarm Optimization, Indianapolis, IN: Purdue School of Engineering and Technology, IUPUI, 2001.

[6] M. Clerc, "The Swarm and the Queen: Towards a Deterministic and Adaptive Particle Swarm Optimization," Proc. Congress on Evolutionary Computation, Washington, DC, 1999.

[7] A. Salman, K. Mehrotra, and C. K. Mohan, "Adaptive Linkage Crossover," *Evolutionary Computation*, vol. 8, pp. 341-370, 2000.