A New Discrete Particle Swarm Optimization Algorithm

Shane Strasser, Rollie Goodman
Montana State University
Dept. of Computer Science
Bozeman MT, 59717-3880
{shane.strasser, rollie.goodman}
@msu.montana.edu

John Sheppard
Montana State University
Dept. of Computer Science
Bozeman MT, 59717-3880
john.sheppard
@montana.edu

Stephyn Butcher
Johns Hopkins University
Dept. of Computer Science
Baltimore, MD 21218-2682
steve.butcher@jhu.edu

ABSTRACT

Particle Swarm Optimization (PSO) has been shown to perform very well on a wide range of optimization problems. One of the drawbacks to PSO is that the base algorithm assumes continuous variables. In this paper, we present a version of PSO that is able to optimize over discrete variables. This new PSO algorithm, which we call Integer and Categorical PSO (ICPSO), incorporates ideas from Estimation of Distribution Algorithms (EDAs) in that particles represent probability distributions rather than solution values, and the PSO update modifies the probability distributions. In this paper, we describe our new algorithm and compare its performance against other discrete PSO algorithms. In our experiments, we demonstrate that our algorithm outperforms comparable methods on both discrete benchmark functions and NK landscapes, a mathematical framework that generates tunable fitness landscapes for evaluating EAs.

Keywords

Particle Swarm Optimization, Discrete Optimization, Categorical Optimization

1. INTRODUCTION

Discrete optimization problems, such as feature selection or inference in Bayesian networks, represent an important and challenging set of problems. These differ from continuous problems in that each variable can take on only a finite number of states [6]. An example is integer problems, where variables are restricted to a set of integer values. For such problems, there exists a relationship between neighboring values. More generally, there is an implicit ordering in the integers: integers with a larger difference between them are considered to be further apart.

While integer problems are a subset of discrete problems, there are other types. For example, in abductive inference for Bayesian networks, the goal is to find the set of states that best explains a set of observations. Here, there may not exist a direct relationship or gradient between neighboring states. For example, say the set of states is the emotions Sadness, Fear, Anger, Joy, and Disgust. While these states may be represented with integers during optimization, there is no real ordered relationship between the values of this *encoding*. We refer to such problems as categorical optimization problems.

Particle Swarm Optimization (PSO) is a highly customizable, yet relatively simple search algorithm applicable to a wide variety of optimization problems. However, the original PSO algorithm is unable to handle discrete problems, such as the ones discussed above, as its velocity update requires continuous solution values [12, 16, 17]. Currently, there are several extensions to the PSO algorithm that allow discrete solution values, though the definition of "discrete" varies widely between applications and algorithms. In this paper, we formally define a class of discrete problems and propose a new PSO algorithm, called Integer and Categorical PSO (ICPSO), designed for this set of problems. We then compare ICPSO to other discrete PSO variants proposed in the literature.

The goal of our algorithm is to keep the extensions to continuous PSO as simple as possible and preserve much of the original semantics, while also addressing some of the potential pitfalls of other discrete PSO algorithms. To achieve this, we alter the representation of the particle's position so that each attribute in a particle is a distribution over its possible values rather than a value itself. This is similar to Estimation of Distribution Algorithms (EDAs) where a set of fit individuals is used to generate a distribution vector that can then generate fitter solutions [14]. ICPSO differs from EDAs in that the algorithm has several distribution vectors that are updated using the PSO update equations.

For ICPSO, evaluating a particle becomes the task of sampling a candidate solution from these distributions and then calculating its fitness. ICPSO also allows us to use the original PSO update equations and avoids problems associated with an implicit ordering of the possible solution values. Additionally, ICPSO modifies the global and local best solutions' distributions whenever a global best sample is produced. This serves to bias distributions toward the best sample they have produced while still allowing exploration of the search space.

The remainder of this paper proceeds as follows. In Section 2, we formally define ICPSO. Section 3 presents a more detailed overview of other discrete PSO algorithms, to which

GECCO '16, July 20-24, 2016, Denver, CO, USA

DOI: http://dx.doi.org/10.1145/2908812.2908935

we empirically compare ICPSO in Section 4. Finally, Section 5 presents our conclusions and future work.

2. ICPSO

This section briefly introduces the original continuous PSO algorithm and the necessary definitions and modifications for our discrete algorithm. Though our experiments do include maximization problems, for the definitions we are assuming a minimization problem where $f(x)$ denotes the fitness function for the optimization.

2.1 Traditional PSO

In traditional PSO, as introduced in [10], a particle p's position in the search space, $\mathbf{X_p} = \{X_{p,1}, X_{p,2}, \ldots, X_{p,N}\}$, directly represents a candidate solution. The particle "flies" through the search space according to its velocity vector $\mathbf{V_p} = \{V_{p,1}, V_{p,2}, \ldots, V_{p,N}\}$, which is updated at each iteration. In the simplest case, this velocity and position updates are

$$\mathbf{V_p} = \omega\mathbf{V_p} + U(0,\phi_1) \otimes (\mathbf{pBest} - \mathbf{X_p}) + U(0,\phi_2) \otimes (\mathbf{gBest} - \mathbf{X_p}) \quad (1)$$

$$\mathbf{X_p} = \mathbf{X_p} + \mathbf{V_p} \quad (2)$$

where each operator is performed component-wise over each variable in the vector and $U(0,\phi_1)$ and $U(0,\phi_2)$ are uniformly distributed random numbers between 0 and ϕ_1 and 0 and ϕ_2. The vectors **pBest** and **gBest** represent, respectively, the best position in the search space this particle has ever seen, and the best position in the search space any particle in the swarm has ever seen. This has the effect of pulling the particle in three directions: the direction it was previously going, the direction of its personal best, and the direction of the global best. By tuning ω, ϕ_1, and ϕ_2, the user may control the relative effects of these three terms – known as inertia, the cognitive component, and the social component – to tune the particle's behavior. This updated velocity is added to the particle's position vector at the current iteration, moving the particle through the search space.

One limitation to this algorithm is that it assumes continuous state variables. If solution variables must take on discrete values, this representation is no longer appropriate. In the next section, we suggest an alternate PSO algorithm that produces discrete-valued solutions without deviating significantly from the traditional update equations.

2.2 Our Approach

We propose an alternative position representation that supports discrete-valued solutions. We first describe the particle representation used in this PSO variant, followed by the necessary changes to the update equations. Finally, we give a modified fitness evaluation procedure and explain how to set the personal and global best vectors.

2.2.1 Defining the Class of Problems

Across the literature, "discrete" PSO has taken on a number of meanings. Many papers only consider integer problems; however, there are many applications where the discrete solution values are not integers. Similarly, while categorical optimization is a form of discrete optimization, not all discrete optimization is necessarily categorical. Due to this ambiguity in the literature, we propose a more specific definition of discrete optimization problems.

Definition 2.1. Discrete Optimization. A class of problems where an objective function is to be optimized that has variables whose values are limited to finite sets, numerical or categorical, ordered or unordered.

The definition of discrete optimization is often a bit fuzzy because integers are frequently used as an example. This definition recognizes that discrete optimization problems often require integer solutions where the possible values for each value are numerical, discrete, ordered and there is a fitness relationship between adjacent or nearby values. But it also emphasizes that some discrete problems are over variables with categorical values such as the emotion example, {Sadness, Fear, Anger, Joy, Disgust}. In this case, there is not necessarily a fitness relationship between the values Sadness and Fear because the set is unordered. More importantly, we could resort to an integer encoding for this set for representational convenience, just because the integers in the encoding are ordered, this does not mean there is now a fitness relationship between adjacent values in the encoding. For example, we could just as easily have encoded our set of emotions from above: {Sadness, Fear, Anger, Joy, Disgust} as $\{1,2,3,4,5\}$ or $\{3,5,4,2,1\}$.

Traditionally, discrete includes both finite and countably infinite sets. For practical reasons, we limit ourselves to finite sets due to finite computer memory. However, our algorithm is applicable to all discrete problems falling within this definition. We refer to our algorithm as Integer and Categorical PSO (ICPSO) to emphasize that it can be applied to both integer and categorical discrete problems.

2.2.2 Representation

The current position for a particle in ICPSO is a set of probability distributions, one for each dimension of the solution. This differs from other PSO variants, where a particle's position is often a direct representation of the solution values. When optimizing over discrete values, this direct representation creates a problem: namely, there is an assumption that there must be a relationship between neighboring states, and that the arithmetic difference between states must be indicative of distance between them. While certain discrete applications may have a fairly natural way to order the possible states, others do not.

Using our emotions example from before, suppose we have a solution with three attributes and {Sadness, Fear, Anger, Joy, Disgust} is encoded as $\{1,3,2,4,5\}$. Say we have two particles with positions $\mathbf{P_{p_1}} = (1,4,5)$ and $\mathbf{P_{p_2}} = (1,3,5)$, and the global best is at $\mathbf{gBest} = (1,5,5)$. The vector subtraction during the velocity update would imply that $\mathbf{p_1}$ is closer to the global best than $\mathbf{p_2}$ is. However, semantically, one could argue that Anger is closer to Disgust than Joy is; therefore, the variable ordering is not indicative of a meaningful ordering of the states.

ICPSO's particle representation avoids this problem by using probability distributions rather than single values in the position vector. A particle p's position is represented as

$$\mathbf{X}_p = [\mathcal{D}_{p,1}, \mathcal{D}_{p,2}, \ldots, \mathcal{D}_{p,n}]$$

where each $\mathcal{D}_{p,i}$ denotes the probability distribution for variable X_i. In other words, each entry in the particle's position vector is itself comprised of a set of distributions:

$$\mathcal{D}_{p,i} = [d^a_{p,i}, d^b_{p,i}, \ldots, d^k_{p,i}].$$

$d^j_{p,i}$ corresponds to the probability that variable X_i takes on value j for particle p.

A particle's velocity is a vector of n vectors ϕ, one for each variable in the solution, that adjust the particle's probability distributions. Formally, this is represented as:

$$\mathbf{V}_p = [\phi_{p,1}, \phi_{p,2}, \ldots, \phi_{p,n}]$$
$$\phi_{p,i} = [\psi^a_{p,i}, \psi^b_{p,i}, \ldots, \psi^k_{p,i}].$$

where $\psi^j_{p,i}$ is particle p's velocity for variable i in state j. Since these values are continuous, the velocity update equation can be used nearly identically to the update equation in traditional PSO.

2.2.3 Update Equations

The velocity and position update equations are identical to those of traditional PSO, seen in Equations (1) and (2). However, because we are working with distributions instead of real values, the difference and addition operators for the distribution and velocity vectors take on a slightly different meaning. For this reason, we will define them explicitly as follows. The difference operator is defined as a component-wise difference between the two position vectors, i.e. for each variable X_i and value $j \in Vals(X_i)$, $d^j_{(\mathbf{pBest_p}-\mathbf{P_p}),i} = d^j_{pB,i} - d^j_{p,i}$. Here, d^j_{pB} is the personal best position's probability that variable X_i takes value j. The global best equation is identical except $\mathbf{pBest_p}$ is replaced with **gBest** and $d^j_{pB,i}$ with $d^j_{gB,i}$.

The addition of the velocity vector to the position vector is similarly component-wise over each value in the distribution. For each probability for variable X_i and possible value j, the addition is $d^j_{p,i} + \psi^j_{p,i}$.

These operations have the potential to create probabilities that fall outside $[0, 1]$. In order to maintain a valid probability distribution, any value outside this range is mapped to the nearest boundary. The distribution is then normalized to ensure that its values sum to 1.

To evaluate a particle p, its distributions are sampled to create a candidate solution. This sample is denoted

$$\mathbf{S}_p = [s_{p,1}, s_{p,2}, \ldots, s_{p,n}]$$

where $s_{p,j}$ denotes the state of variable X_j.

The fitness function is used to evaluate the sample's fitness, which then is used to evaluate the distribution.

2.2.4 Setting The Best Vectors

When a particle produces a sample that beats the global or local best, we use both the distributions from that particle's position, $\mathbf{P}_p$, and the sample itself, $\mathbf{S}_p$, to update the best values. The goal of this update is to bias the distribution that produced the best sample toward producing similar samples in the future. This is accomplished by reducing the probability, for each variable, of taking on any state except its state in the best sample. Mathematically, for all states $j \in Vals(X_i)$ the global best's probability is updated as

$$d^j_{gB,i} = \begin{cases} \epsilon \times d^j_{p,i} & \text{if } j \neq s_{p,i} \\ d^j_{p,i} + \sum\limits_{\substack{k \in Vals(X_i) \\ \wedge k \neq j}} (1-\epsilon) \times d^k_{p,i} & \text{if } j = s_{p,i} \end{cases}$$

where ϵ, the *scaling factor*, is a user-set parameter that determines the magnitude of the shift in the distribution. We restrict the scaling factor to values in $[0, 1)$. This increases the likelihood of the distribution producing samples similar to the best sample, while inherently maintaining a valid probability distribution. This can be shown as follows:

$$\begin{aligned} \sum_{j \in Vals(X_i)} d^j_{gB,i} &= \sum_{j \in Sv(X_i,k)} \left(d^j_{p,i} \times \epsilon \right) + d^k_{p,i} + \\ &\quad \sum_{j \in Sv(X_i,k)} \left(d^j_{p,i} - d^j_{p,i} \times \epsilon \right) \\ &= \sum_{j \in Sv(X_i,k)} \left(d^j_{p,i} \times \epsilon \right) + d^k_{p,i} + \\ &\quad \sum_{j \in Sv(X_i,k)} \left(d^j_{p,i} \right) - \sum_{j \in Sv(X_i,k)} \left(d^j_{p,i} \times \epsilon \right) \\ &= d^k_{p,i} + \sum_{j \in Sv(X_i,k)} \left(d^j_{p,i} \right) = 1 \end{aligned}$$

where $Sv(X_i, k) = \{j | j \in Vals(X_i) \wedge j \neq k\}$ and $k = s_{p,i}$. The procedure for setting the local best is directly analogous. The global best sample is returned as the solution at the end of optimization.

3. RELATED WORK

In this section, we review other approaches to discrete particle swarm optimization, both binary and multi-valued. This analysis provides the necessary context for our experiments comparing the performance of ICPSO to competing approaches. Additionally, this section serves to highlight the problems, endemic in discrete PSO methods, that our contributions are intended to address.

First, we discuss PSO variants that mostly maintain the core update equations and have only been augmented with a few additional equations to handle discrete problems. These algorithms are the most similar to ours, and thus the bulk of the discussion will be focused here.

Next, we present versions of PSO that have been combined with other optimization algorithms, such as estimation of distribution algorithms (EDA). We include these hybrid algorithms because ICPSO shares some similar ideas, though it differs in implementation. Finally, we discuss other PSO variants applied to specific discrete problems, such as the Traveling Salesman Problem. While these algorithms are only designed for specific applications, we include them in our discussion for the sake of completeness.

3.1 Discrete Variations of PSO

Integer PSO: It is possible to use the original continuous PSO to solve problems with integer-valued solutions by rounding the particle's position at each iteration [10]. We will refer to this algorithm as Integer PSO (IPSO).

IPSO requires a relationship between neighboring states, as the velocity update equation uses subtraction to measure the distance between the particle's current position and the global/local best positions. As such, it is only applicable to certain problems where the states can be ordered or arranged in a way that provides the necessary relationship between neighboring states.

Binary PSO: Another discrete PSO algorithm is Binary PSO (BPSO), originally proposed by Kennedy and Eberhart [12]. BPSO requires that the position vector be a binary representation of candidate solutions. This representation also changes the velocity interpretation: the velocity represents the probability of each variable assuming the value

0 or 1. While the velocity update for BPSO remains unchanged, the position update is modified to take advantage of the new semantics of the velocity vector. After updating the velocity vector, each term in the velocity is mapped into a [0,1] interval using the sigmoid function

$$S_{ig} = \frac{1}{1 + \exp(-V_{i,j})} \tag{3}$$

where $V_{i,j}$ is the value of the jth variable for particle i [12]. Next, a random number is sampled from the normal distribution $X_{i,j} = \mathcal{N}(0, 1)$ and converted to be in $[0, 1]$ by first calculating $S_{i,j} - X_{i,j}$, and then using a unit set function to snap the difference to 0 or 1. This value is then assigned as particle i's current position for variable j. When velocity is high, the position update is more likely to select a value closer to 1 than 0. Conversely, when velocity is low, there is a higher likelihood of selecting 0 [12].

The main limitation of BPSO is that it requires a binary representation. Binary coding is commonly used for binary representation, but has the disadvantage of introducing Hamming cliffs. Hamming cliffs represent situations where adjacent binary-encoded numbers have a large Hamming distance between them, or where two binary-encoded numbers with a very small Hamming distance actually have a large difference in value.

An alternative strategy is to use Gray coding, where the Hamming distance between neighboring values is set to 1, which reduces the Hamming Cliff problem. However, both methods' encoding may overrepresent the problem if the number of states is not a power of 2. This can also increase problem dimensionality, slowing optimization [15].

Veeramachaneni PSO: Veeramachaneni *et al.* developed an extension to binary PSO that relaxes the need for a binary representation of the problem [22]. Throughout the rest of the paper, we will refer to this algorithm as the Veeramachaneni PSO (VPSO). In VPSO, each variable is allowed to assume any of M discrete values. While the velocity update remains unchanged from the binary case, the position update is modified to allow for more than 2 states. After the velocity has been updated, it is mapped into the $[0, M - 1]$ interval by first using a generalized version of the sigmoid function in Equation (3), which is given as

$$S_{i,j} = \frac{M - 1}{1 + \exp(-V_{i,j})}.$$

Next, each particle's position is updated by generating a random number according to the normal distribution $X_{i,j} = \mathcal{N}(S_{i,j}, \sigma \times (M - 1))$ and rounding the result. Then the piecewise function

$$X_{i,j} = \begin{cases} M - 1 & X_{i,j} > M - 1 \\ 0 & X_{i,j} < 0 \\ X_{i,j} & \text{otherwise} \end{cases}$$

is applied to ensure all values fall within $[0, M - 1]$ [22].

While VPSO does extend BPSO so any number base can be used, the algorithm requires a relationship between neighboring states in the range of variable values, just like in IPSO. In [22], the authors demonstrated that VPSO is able to outperform BPSO when mapping continuous variables to quaternary or ternary. However, all experiments contained relationships between neighboring states.

Pugh PSO: The multi-valued PSO extension most similar to ours was introduced by Pugh and Martinoli [17]. We will refer to this variant as PPSO. PPSO, like ICPSO, uses a probabilistic interpretation of a particle and evaluates fitness stochastically by generating a sample solution.

The position vector, however, does not represent a valid probability distribution explicitly in PPSO. When generating a sample, each element in the position vector has a sigmoid transformation applied to each of its terms. Then, the probability of the sample solution's jth element taking on value k is the kth term of the position vector's jth element divided by the weighted sum of all the elements in that term. This allows the solution element to take on any value from 0 to a user-specified n. This fitness evaluation involves several more steps than our implementation of discrete PSO.

An adjustment must also be applied after each modification of the particle's values in order for all of the particles to share a common reference frame [17]. To achieve this adjustment, a value c_{ij} is subtracted from each value of particle i, element j's vector of values. This c_{ij} is calculated such that all values of the vector, when mapped to the sigmoid function, sum to 1. The resulting equation is solved for c via an approximate root-finding method to produce an adjustment for each value in each element of the position vector. To address the noisy fitness evaluation, the algorithm also reevaluates best particles at each iteration, averaging fitness over particles' lifetimes. Unlike in ICPSO, the sample's value is not used when setting the global or local best positions.

Angle Modulated PSO: Angle Modulated PSO, also known as AMPSO, reduces a high-dimensional binary search space into a smaller continuous search space using an angle-modulation-based method, thus reducing the number of parameters to be optimized [16]. This speeds up optimization while potentially improving performance. The approach uses the angle modulation equation, which is given as

$$g(x) = \sin(2\pi(x - a) \times b \times \cos(A)) + d$$

where $A = 2\pi \times c(x - a)$ and x is a single input value.

AMPSO first optimizes over the parameters a, b, c, and d in the angle modulation equation. Next, for each variable, the algorithm generates k evenly-spaced values, where k is the number of bits needed to represent every state in the discrete problem. These k values are then transformed into a bit string by converting positive values to 1 and negative values to 0.

The novelty of AMPSO is more related to the transformation of the search space than the PSO implementation itself. Additionally, this approach requires a binary representation of the problem, which has similar limitations to BPSO.

3.2 Hybrid Algorithms

Another related class of algorithms combines Estimation of Distribution Algorithms (EDA) and PSO to create an EDA-PSO hybrid [1, 5, 23]. These approaches either use EDA to help guide the movement of particles in PSO, or use PSO to generate or update individuals generated by an EDA. Our work differs from these hybrid approaches in that it retains the core PSO update equations by modifying the particle representation to fit discrete problems. Additionally, many of these hybrid algorithms are designed to operate on continuous problems, while ICPSO is specifically designed for discrete optimization.

In the work by El-Abd and Kamel, the authors propose a hybrid algorithm in which a particle is either updated according to the PSO update equations or replaced with a new

individual sampled from the estimated distribution [5]. In another example of a hybrid algorithm, Zhou *et al.* developed an algorithm called Discrete Estimation of Distribution Particle Swarm Optimization (DEDPSO), in which the local best positions from all individuals are used to update the distribution vector. This update distribution vector is then sampled to update the existing individuals [23]. The algorithm is designed to operate on binary vectors.

In [18], Reynolds *et al.* first generate a set of individuals using EDA and insert them into a PSO swarm. PSO runs for a set number of iterations and then uses the updated positions to generate new individuals. Those individuals are then added back to the original set of individuals from EDA. The set of individuals from both algorithms are then used to update the probability distribution, and the process is repeated. This is similar to the work by Bengoetxea and Larrañaga [1]. Their algorithm generates individuals independently using EDA and PSO and uses the generated individuals from both algorithms to update the distribution.

Kulkarni and Venayagamoorthy use both EDA and PSO when updating an individual [13]. An individual is first updated according to the PSO update equations, and a new individual is generated using EDA. Of these, the individual with the best fitness is kept in the swarm [13]. Santucci and Milani take a different approach to hybridization by using PSO within an EDA framework [20]. This is done by replacing the PSO update equations with those found in EDA. Then each variable in an individual is updated using EDA. After all individuals have been updated, the distribution is updated using the new positions of the individuals [20].

3.3 Other Approaches

In addition to the hybrid and discrete PSO algorithms, there are several PSO variants tailored to specific discrete problems, such as the Traveling Salesman Problem (TSP). In these cases, each application requires a specific mapping or transformation of the problem [2]. For example, Clerk uses PSO to solve TSP by representing the particle as a path through all nodes [3]. With this representation, the velocity is a set of changes to be made to the path. The addition and subtraction operators are then re-defined to fit the modified semantics of the optimization [3].

Sha *et al.* propose a PSO algorithm to solve the job scheduling problem [21]. Each particle represents a matrix, where each element is the priority of a job on a machine. The velocity represents a swap operator of a job to a different machine. To evaluate a solution, the matrix is decoded into a schedule using Giffler and Thompson's heuristic [8]. This schedule is then evaluated for fitness.

Other approaches use a variation of integer PSO by rounding continuous values to integer values during fitness evaluation. This is similar to the work done by Salman *et al.*, where the authors applied PSO to the task assignment problem [19]. Each particle's position represents a matrix that contains assignments of a task to a machine or processor. The velocity update remains unchanged, but is also represented by a matrix. Particle evaluation is done after rounding values in the matrix [19].

Hela and Abdelbar also used a matrix representation in using PSO to solve the quadratic assignment problem [9]. In that work, the velocity is represented by a matrix where an element (i, j) represents the likelihood of variable i taking on value j. The position is then updated by probabilistically selecting an element for variable i, using roulette wheel selection, based on the values in row i of the velocity matrix. To update the velocity, the position array is expanded to a binary position matrix where $(i, j) = 1$ if variable i is set to value j. In some ways, this type of matrix-based representation is similar to the one used in ICPSO. However, the differ in that our position matrix serves the same function as their velocity matrix and our sample position corresponds to their algorithm's position array.

A more recent algorithm for combinatorial optimization problems uses what is called set-based PSO [2]. Here, each individual represents a subset of values out of a universal set, and the velocity represents the probability of an element being selected for inclusion in the set. To fit the updated position and velocity semantics, the authors define new set-theoretic velocity and position update equations [2].

4. EXPERIMENTS

We compare ICPSO against the algorithms from Section 3.1, as those were the closest in approach and interpretation. Specifically, we compared to the Angle Modulated PSO (AMPSO), Binary PSO (BPSO), Binary PSO using Gray coding (BGPSO), Integer PSO (IPSO), the PSO proposed by Pugh and Martinoli (PPS), and the PSO proposed by Veeramachaneni *et al.* (VPSO).

4.1 Design

We compare the algorithms on a set of benchmark functions (namely, Ackley, Griewank, Rastrigin, Rosenbrock, and Sphere) using the same function ranges as presented in the appendix of [4]. Normally, these are studied as continuous functions to be optimized; however, we modify them to allow for both integer and categorical (discrete) optimization. For the integer discrete optimization problem, we restrict the states of the variables to integer values. This permits adjacent values to correlate with their fitness as defined by the original function. For categorical problems, and unique to our experimental design within the discrete PSO literature, we break the relationship between the adjacent numerical values and their fitness by mapping the integer values to a randomly chosen ("shuffled") integer encoding. For example, the state values of some variable x, $\{1, 2, 3, 4, 5\}$, might be shuffled to the encoding $\{4, 2, 1, 3, 5\}$.

We tested all discrete algorithms on both shuffled (categorical) and unshuffled (integer) versions. Functions were restricted to 10 dimensions and 10 states for each dimension. For the shuffled versions, we generated 30 different shuffles and ran each algorithm 30 times on the each function. Algorithms were also run 30 times on each of the unshuffled problems.

We also tested the algorithms on NK landscapes, which are a mathematical framework that generates tunable fitness landscapes. An NK landscape model contains two parameters, N and K, that control the overall size of the landscape and the structure or amount of interaction between each dimension [11]. Usually, NK landscapes are binary strings. We, however, used a generalized version that allows for integer strings, where each variable can take on D different values. For our experiments, we used an NK-landscape with $N = 10$ and $K = 2$. We varied the number of states, D, for each dimension to values 2, 4, 6, 8, 10, 15, and 20. For each set of NK landscape parameters, we generated 30 different landscapes and ran each algorithm 30 times per landscape.

All PSO variants used the same set of parameters, in order to make comparisons as consistent as possible. Parameters ϕ_1 and ϕ_2 were set to 1.49618, and $\omega = 0.729$, which has been found to encourage convergent trajectories [4]. Each algorithm used a swarm of size 5, and terminated once the global best did not change after 50 iterations. This is due to the recommendations of [7], which demonstrated that a large swarm may, counterintuitively, have difficulty exploring the search space. For our approach, we set the scaling factor ϵ to 0.75, and in the VPSO we set σ to the authors' recommended value of 0.2. All algorithms randomly initialized velocity and position vectors. Significance testing was done using a paired Student t-Test with $\alpha = 0.05$.

4.2 Results

The results from the test functions, reported as average solution fitness, are shown in Table 1. Standard error is shown in parentheses. Bold values indicate algorithms that statistically significantly outperformed *all* other algorithms.

On the unshuffled functions, IPSO significantly performed the best. The next best-performing algorithm was ICPSO. Even though it is not shown in the table, ICPSO significantly outperformed all other algorithms except IPSO. AMPSO, BPSO, and BGPSO all had comparable performance. VPSO was comparable with AMPSO, BPSO, and BGPSO and PPSO generally outperformed VPSO.

For the shuffled problems, ICPSO statistically performed the best, with IPSO as the runner-up. The rest of the algorithms have roughly the same performance, and are function-dependent as to which method performs the best.

Comparing the each algorithm across the regular and shuffled problems, we found that ICPSO had the smallest change in performance. In some cases the performance on the unshuffled problems was better than the shuffled; however, this was not always the case. Meanwhile, IPSO consistently had poorer performance on the shuffled problems. AMPSO usually performed better on the unshuffled problems, except on the Rastrigin function. BPSO and BGPSO both performed better on the shuffled functions than unshuffled.

Table 2 contains the results of the PSO variants on maximizing NK landscapes. The far left column indicates the number of states per dimension. On the NK landscapes, ICPSO almost always demonstrated the best performance significantly, and was only outperformed on binary strings ($D = 2$). In the binary case, BPSO, BGPSO, and VPSO significantly performed the best. Many of the other algorithms, such as AMPSO, BPSO, and BGPSO only varied slightly for $D = 4$ to 20. However, ICPSO had among the highest variance in performance between different D values.

The fitness curves for all of the PSO algorithms on the sphere and shuffled sphere problems are shown in Figures 1 and 2. For ease of reading, the results in each case have been split between two graphs with the same scale. The X-axis is the number of iterations, while the Y axis is the **gBest** fitness averaged over 30 runs. For these experiments, we ran all algorithms for 200 iterations.

4.3 Analysis

Based on our results, ICPSO is generally more robust than the other approaches, as demonstrated by its consistent performance on the shuffled and unshuffled functions. We believe this is because the particles represent distributions instead of candidate solutions, and thus do not rely on having a gradient or relationship between neighboring states.

Table 1: Results of discrete PSO algorithms on minimizing benchmark functions.

	Ackleys	Shuffled Ackleys
AMPSO	4.83E00(5.08E-01)	5.35E00(9.05E-02)
BPSO	6.86E00(1.08E-01)	5.32E00(1.36E-01)
BGPSO	6.70E00(7.97E-02)	5.42E00(1.05E-01)
ICPSO	3.23E00(1.30E-01)	**3.18E00(1.44E-01)**
IPSO	**1.23E00(2.05E-01)**	4.43E00(1.60E-01)
PPSO	4.35E00(1.70E-01)	4.13E00(1.56E-01)
VPSO	5.98E00(1.01E-01)	5.14E00(1.04E-01)

	Griewank	Shuffled Griewank
AMPSO	8.19E-01(5.48E-02)	8.64E-01(1.52E-02)
BPSO	9.60E-01(4.96E-03)	8.22E-01(1.44E-02)
BGPSO	9.48E-01(6.46E-03)	8.50E-01(1.27E-02)
ICPSO	4.44E-01(2.58E-02)	**5.15E-01(3.32E-02)**
IPSO	**2.51E-01(4.01E-02)**	7.51E-01(2.26E-02)
PPSO	8.28E-01(1.84E-02)	8.67E-01(1.50E-02)
VPSO	9.20E-01(8.20E-03)	7.59E-01(1.75E-02)

	Rastrigin	Shuffled Rastrigin
AMPSO	2.14E04(3.43E03)	1.29E04(3.72E03)
BPSO	3.94E04(1.65E03)	6.80E03(3.88E02)
BGPSO	3.53E04(2.24E03)	7.18E03(3.41E02)
ICPSO	2.14E03(4.79E02)	**1.92E03(2.73E02)**
IPSO	**4.37E02(1.29E02)**	7.10E03(8.00E02)
PPSO	8.84E03(1.26E03)	9.05E03(1.34E03)
VPSO	1.17E04(1.31E03)	4.32E03(2.46E02)

	Rosenbrock	Shuffled Rosenbrock
AMPSO	1.39E04(3.60E03)	1.82E4(1.98E03)
BPSO	3.38E04(2.19E03)	2.66E4(1.83E03)
BGPSO	3.32E04(2.05E03)	2.53E4(1.88E03)
ICPSO	2.04E03(3.24E02)	**1.75E3(2.17E02)**
IPSO	**2.89E02(6.39E01)**	5.04E3(9.35E02)
PPSO	8.47E03(1.02E03)	9.31E3(1.33E03)
VPSO	1.22E04(1.36E03)	2.42E4(1.63E03)

	Sphere	Shuffled Sphere
AMPSO	2.39E01(3.59E00)	2.99E01(1.35E00)
BPSO	4.50E01(1.78E00)	3.06E01(1.28E00)
BGPSO	4.63E01(1.54E00)	3.15E01(1.13E00)
ICPSO	8.80E00(7.94E-01)	**7.33E00(6.46E-01)**
IPSO	**2.07E00(9.06E-01)**	1.35E01(6.98E-01)
PPSO	1.59E01(1.46E00)	1.68E01(1.26E00)
VPSO	3.05E01(1.47E00)	2.74E01(1.17E00)

While PPSO uses a very similar particle representation to ours, ICPSO always outperformed PPSO. We believe that this is due in part to how we set and bias the local and global bests. ICPSO uses the knowledge that if particular sample has high fitness, more exploration should likely be performed around the sample. Another benefit to our approach is that we avoid the added complexity that PPSO incurs due to the approximate methods it requires to shift position values. ICPSO instead treats each variable as a probability distribution and normalizes after the position update.

As the number of states varied in NK landscapes, ICPSO had the highest variance in terms of its performance. This could be because it uses samples to set the local and global best vectors, which may make ICPSO more sensitive to bias

Table 2: Results of different discrete PSO algorithms on maximizing NK landscapes.

	AMPSO	BPSO	BGPSO	ICPSO	IPSO	PPSO	VPSO
D = 2	7.16(0.02)	**7.40(0.02)**	**7.39(0.02)**	7.23(0.02)	6.19(0.02)	7.27(0.02)	**7.40(0.02)**
D = 4	7.44(0.02)	7.61(0.01)	7.61(0.01)	**7.97(0.01)**	6.48(0.02)	7.73(0.01)	7.51(0.01)
D = 6	7.49(0.02)	7.64(0.01)	7.63(0.01)	**8.14(0.01)**	6.64(0.02)	7.79(0.01)	7.65(0.01)
D = 8	7.47(0.02)	7.64(0.01)	7.64(0.01)	**8.09(0.01)**	6.61(0.02)	7.80(0.01)	7.65(0.01)
D = 10	7.44(0.02)	7.62(0.01)	7.63(0.01)	**8.11(0.01)**	6.67(0.02)	7.78(0.01)	7.66(0.01)
D = 15	7.44(0.02)	7.64(0.01)	7.63(0.01)	**8.02(0.01)**	6.70(0.02)	7.75(0.01)	7.66(0.01)
D = 20	7.48(0.02)	7.62(0.01)	7.61(0.01)	**7.98(0.02)**	6.67(0.02)	7.72(0.01)	7.64(0.01)

associated with sampling. Additionally, this could be caused by under-sampling the distribution.

Results also suggest that IPSO performs the best if a gradient exists between the states for a variable. If no such gradient exists, then IPSO suffers the largest drop in performance. This implies that ICPSO may be a more natural choice for categorical optimization, as its performance is unaffected by the order or interpretation of the discrete values.

The fitness curves for the sphere problems show that ICPSO has a steep initial curve. However, there appear to be decreasing returns to fitness after about 100 iterations. IPSO has a similar trend on the unshuffled sphere problem shown in the left graph in Figure 1. Some other approaches, such as PPSO and VPSO, converge at a comparatively slow pace.

Analysis of the fitness curves in Figures 1 and 2 suggests that ICPSO might also be useful in applications where only a limited number of iterations may be run. Since ICPSO generally has a sharp initial fitness gain and levels out within relatively few iterations, it can return a good solution sooner than many of the competing algorithms.

5. CONCLUSIONS AND FUTURE WORK

Several conclusions can be drawn from this work. First, ICPSO largely avoids pitfalls associated with assuming relationships between neighboring states in discrete problems, as demonstrated by its consistent performance across the shuffled and unshuffled sphere problems. It also requires few modifications to the PSO update equations. Since the probabilities are scalars, operators can be applied directly to the values. ICPSO also frequently outperforms related methods on both maximizing NK landscapes and optimizing well-known benchmark functions. The fitness curves suggest that ICPSO is able to reach a good solution sooner than competing algorithms, often reaching decreasing returns by 100 iterations on the test problems.

ICPSO is suited for categorical as well as integer problems, and retains many of the original features of continuous PSO. It also generally converges to a good solution quickly in the applications tested. However, it is still sometimes outperformed by other discrete PSO approaches, such as IPSO. We also plan to explore the conditions under which this is likely to occur, including when applied to combinatorial optimization problems (e.g., graph coloring).

Several extensions could be made to ICPSO. For example, in real-world problems, solution values are not always independent as they are assumed to be here. Extending ICPSO to incorporate constraints and dependencies, possibly using a probabilistic graphical model, could improve performance for some applications. Finally, we want to investigate why ICPSO is sensitive to the number of states per variable.

6. REFERENCES

[1] E. Bengoetxea and P. Larrañaga. EDA-PSO: a hybrid paradigm combining estimation of distribution algorithms and particle swarm optimization. In *Swarm Intelligence*, pages 416–423. Springer, 2010.

[2] W.-N. Chen, J. Zhang, H. S. Chung, W.-L. Zhong, W.-G. Wu, and Y.-H. Shi. A novel set-based particle swarm optimization method for discrete optimization problems. *IEEE Transactions on Evolutionary Computation*, 14(2):278–300, 2010.

[3] M. Clerc. Discrete particle swarm optimization, illustrated by the traveling salesman problem. In *New optimization techniques in engineering*, pages 219–239. Springer, 2004.

[4] R. C. Eberhart and Y. Shi. Comparing inertia weights and constriction factors in particle swarm optimization. In *Proceedings of the 2000 Congress on Evolutionary Computation*, volume 1, pages 84–88. IEEE, 2000.

[5] M. El-Abd and M. S. Kamel. Black-box optimization benchmarking for noiseless function testbed using an EDA and PSO hybrid. In *Proceedings of the 11th Annual Conference Companion on Genetic and Evolutionary Computation Conference: Late Breaking Papers*, pages 2263–2268, 2009.

[6] A. P. Engelbrecht. *Computational intelligence: An introduction*. John Wiley & Sons, 2007.

[7] A. P. Engelbrecht. Fitness function evaluations: A fair stopping condition? In *Proceedings of the IEEE Swarm Intelligence Symposium (SIS)*, pages 1–8. IEEE, 2014.

[8] B. Giffler and G. L. Thompson. Algorithms for solving production-scheduling problems. *Operations research*, 8(4):487–503, 1960.

[9] A. M. Helal and A. M. Abdelbar. Incorporating domain-specific heuristics in a particle swarm optimization approach to the quadratic assignment problem. *Memetic Computing*, 6(4):241–254, 2014.

[10] K. James and E. Russell. Particle swarm optimization. In *Proceedings of the IEEE International Conference on Neural Networks*, pages 1942–1948, 1995.

[11] S. A. Kauffman. *The origins of order: Self-organization and selection in evolution*. Oxford university press, 1993.

[12] J. Kennedy and R. C. Eberhart. A discrete binary version of the particle swarm algorithm. In *IEEE International Conference on Systems, Man, and Cybernetics. Computational Cybernetics and Simulation*, volume 5, pages 4104–4108, 1997.

[13] R. V. Kulkarni and G. K. Venayagamoorthy. An

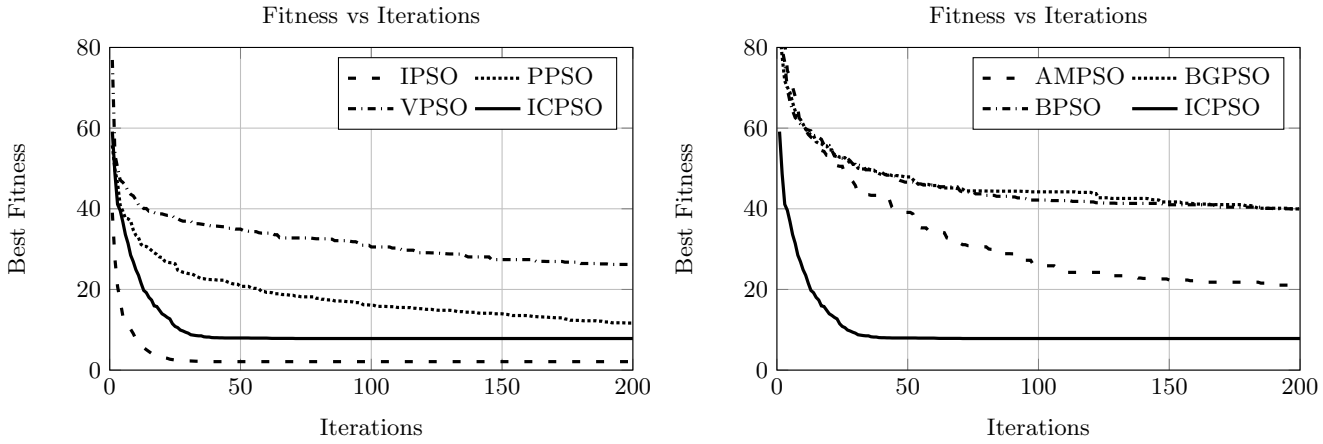

Figure 1: Fitness curves for minimizing the discrete sphere function.

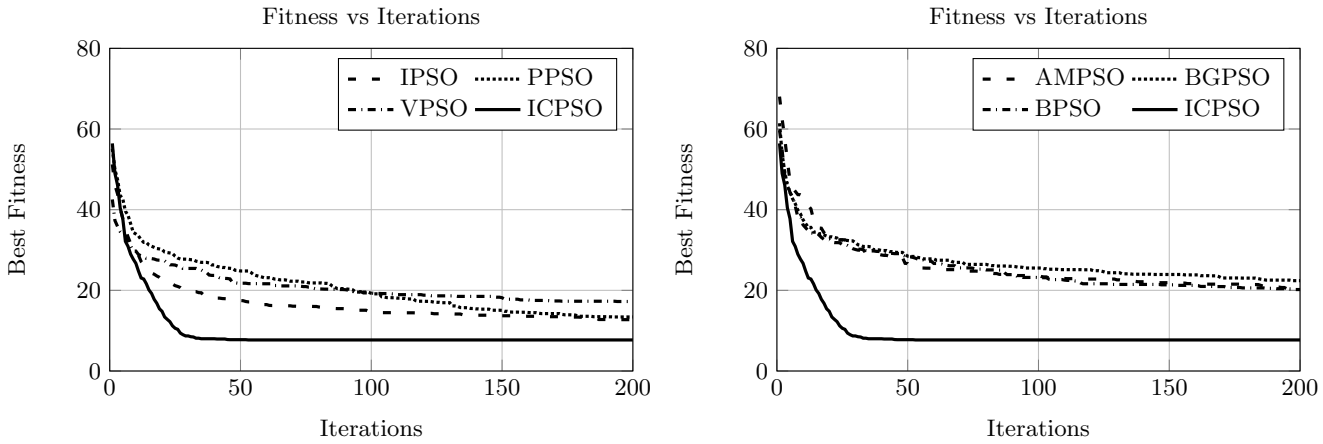

Figure 2: Fitness curves for minimizing the shuffled discrete sphere function.

estimation of distribution improved particle swarm optimization algorithm. In *3rd International Conference on Intelligent Sensors, Sensor Networks and Information (ISSNIP)*, pages 539–544, 2007.

[14] P. Larranaga and J. A. Lozano. *Estimation of distribution algorithms: A new tool for evolutionary computation*, volume 2. Springer Science & Business Media, 2002.

[15] Z. Michalewicz. *Genetic algorithms + data structures = evolution programs*. Springer Science & Business Media, 1996.

[16] G. Pampara, N. Franken, and A. P. Engelbrecht. Combining particle swarm optimisation with angle modulation to solve binary problems. In *IEEE Congress on Evolutionary Computation*, volume 1, pages 89–96, 2005.

[17] J. Pugh and A. Martinoli. Discrete multi-valued particle swarm optimization. In *Proceedings of IEEE Swarm Intelligence Symposium (SIS)*, pages 103–110, 2006.

[18] A. P. Reynolds, A. Abdollahzadeh, D. W. Corne, M. Christie, B. Davies, and G. Williams. A parallel BOA-PSO hybrid algorithm for history matching. In *IEEE Congress on Evolutionary Computation (CEC)*, pages 894–901, 2011.

[19] A. Salman, I. Ahmad, and S. Al-Madani. Particle swarm optimization for task assignment problem. *Microprocessors and Microsystems*, 26(8):363–371, 2002.

[20] V. Santucci and A. Milani. Particle swarm optimization in the EDA framework. In *Soft Computing in Industrial Applications*, pages 87–96. Springer, 2011.

[21] D. Sha and C.-Y. Hsu. A hybrid particle swarm optimization for job shop scheduling problem. *Computers & Industrial Engineering*, 51(4):791–808, 2006.

[22] K. Veeramachaneni, L. Osadciw, and G. Kamath. Probabilistically driven particle swarms for optimization of multi valued discrete problems: Design and analysis. In *Proceedings of the IEEE Swarm Intelligence Symposium (SIS)*, pages 141–149, 2007.

[23] Y. Zhou, J. Wang, and J. Yin. A discrete estimation of distribution particle swarm optimization for combinatorial optimization problems. In *Third International Conference on Natural Computation (ICNC)*, volume 4, pages 80–84, 2007.