MICPSO: A Method for Incorporating Dependencies into Discrete Particle Swarm Optimization

Rollie Goodman, Monica Thornton, Shane Strasser, and John W. Sheppard
Montana State University,
Bozeman, MT 59717 USA
{rollie.goodman, monica.thornton, shane.strasser, john.sheppard}@montana.edu

***Abstract*—In this work, we present an extension to the recently developed Integer and Categorical Particle Swarm Optimization (ICPSO), which we refer to as Markovian ICPSO (MICPSO). MICPSO uses a Markov network to represent a particle's position, thus allowing each particle to incorporate information about dependencies between solution variables. In this work, we compare MICPSO to ICPSO, Integer PSO (IPSO), an Estimation of Distribution Algorithm called Markovianity-Based Optimization Algorithm (MOA), and a hillclimber on a set of benchmark vertex coloring problems. We find that MICPSO significantly outperforms all alternatives on all problems tested.**

I. Introduction

Integer and Categorical Particle Swarm Optimization (ICPSO) is a new discrete particle swarm optimization algorithm that is designed to handle both integer and categorical state variables [1]. This is achieved by representing the particle's position as a set of probability distributions, one per variable, over the possible solution values. Solution values are produced by sampling from these distributions.

ICPSO has been shown to perform well compared to other discrete PSO variants, particularly on problems where there is no natural ordering to solution values. However, it is limited by the fact that the algorithm assumes that variables in the solution are independent of each other. This can cause issues with some constrained optimization problems and many real-world optimization problems where dependencies exist between solution values. In such situations, the algorithm must work around these dependencies, for example by "fixing" the generated solutions or penalizing the fitness of infeasible solutions. Fixing samples creates possible convergence problems for ICPSO, as the fitness of the samples may not accurately reflect the average fitness of the distributions that generated them. Penalizing infeasible solutions is not always practical, particularly if there are very few feasible solutions compared to infeasible ones. In general, methods similar to ICPSO will fail to converge when interactions between problem variables are not accounted for [2]. This limits the class of optimization problems to which ICPSO can be expected to perform well.

In this work, we present an extension to ICPSO that addresses this limitation. Rather than using separate distributions for each state variable, as ICPSO does, our proposed algorithm utilizes localized joint probabilities between pairs of state variables to represent dependencies explicitly. This involves encoding a probabilistic graphical model (in this case, a Markov network) into the particle structure. Doing so allows the algorithm to avoid the problems of fixing or penalizing solutions by integrating knowledge of the dependencies and constraints into the sampling process directly. We hypothesize that this will be an improvement upon ICPSO's performance on constrained optimization problems.

II. Background and Related Literature

In this section, we provide an overview of the related literature, starting with Particle Swarm Optimization (PSO). We then provide background on Markov networks and discuss a subset of Estimation of Distribution Algorithms (EDAs) that use Markov networks to model the population.

A. Particle Swarm Optimization

1) PSO: In the original PSO algorithm, a particle $\mathbf{p}$'s position in the search space, $\mathbf{x_p}$, directly represents a candidate solution to the optimization problem [3]. The particle moves through the search space according to its velocity vector $\mathbf{v_p} = \{v_{p,1}, v_{p,2}, \ldots, v_{p,N}\}$. At each iteration, the fitness of the current position is evaluated, and the velocity vector is updated and added to the position vector in order to determine the particle's new position. In the simplest case, the velocity and position updates are

$$\begin{aligned}\mathbf{v_p} &= \omega\mathbf{v_p} + U(0,\phi_1) \otimes (\mathbf{pBest} - \mathbf{x_p}) \\ &\quad + U(0,\phi_2) \otimes (\mathbf{gBest} - \mathbf{x_p}) \\ \mathbf{x_p} &= \mathbf{x_p} + \mathbf{v_p}\end{aligned}$$

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 $\mathbf{pBest}$ and $\mathbf{gBest}$ 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 simultaneously pulls 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 can control the relative effects of these three terms, known as inertia, the cognitive component, and the social component, to tune the particle's

behavior. After a termination criterion is reached, often a set number of iterations, the global best is returned.

2) IPSO: Integer PSO (IPSO) is a simple extension to the continuous PSO algorithm that allows it to solve problems with integer-valued solution variables [3]. IPSO uses the same position and velocity updates as the original PSO algorithm; however, after the velocity is updated and added to the position, each element of the position vector is rounded to the nearest integer value.

3) Veeramachaneni PSO: Veeramachaneni *et al.* developed an extension to binary PSO that relaxes the need for a binary representation of the problem, which we refer to as Veeramachaneni PSO (VPSO) [4]. In VPSO, each variable is allowed to assume M discrete values. 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, 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 $[0, M-1]$ [4].

4) ICPSO: ICPSO was developed to address a major pitfall of PSO: the original PSO algorithm cannot handle discrete problems, and the existing discrete PSO variants were not appropriate for truly categorical problems because their update equations semantically require a numerical relationship or gradient between the states of the variables [1]. To overcome this limitation, ICPSO represents a particle's position not as a solution itself, but as a set of probability distributions over possible solution values. For a particle $\mathbf{p}$, its position can be 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. Each entry in the particle's position vector is itself comprised of a set of distributions $\mathcal{D}_{p,i} = [d^a_{p,i}, d^b_{p,i}, \ldots, d^k_{p,i}]$, where $d^j_{p,i}$ corresponds to the probability that variable x_i takes on value j for particle $\mathbf{p}$. This allows the same velocity update equation to be used as in traditional PSO, though the interpretation changes: the update is now an adjustment to the particle's probability distributions.

With this representation, a particle's fitness is evaluated by sampling from this distribution and then calculating the fitness of that sample. Multiple samples may be drawn at each iteration to give a better idea of average fitness. When a sample $\mathbf{S}$ is found that beats the local or global best, the best is updated and the distribution is biased toward producing similar samples in the future by updating the entries in the position vector as follows:

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

This has the effect of biasing the particle toward producing samples similar to $\mathbf{S}$ in the future, while also automatically maintaining a valid probability distribution. As with regular PSO, the global best solution – in this case, the best sample, which was generated by the distributions from the last **gBest** particle found – is returned at the end of the optimization.

5) Pugh PSO: Another multi-valued PSO extension was introduced by Pugh and Martinoli [5]. We refer to this variant as Pugh PSO (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, a sigmoid transformation is applied to each term in each element of the position vector. 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. An adjustment must also be applied after each modification of the particle's values so that all of the particles share a common reference frame [5]. 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 reevaluates best particles at each iteration, averaging fitness over particles' lifetimes.

B. Markov Networks

Markov networks are probabilistic graphical models that provide a method of representing complex joint distributions compactly by modeling the local interactions between random variables [6].

1) Interpretation: The nodes in a Markov network's graph represent the variables in the problem, and the edges, all of which are undirected, represent probabilistic interaction between variables [7]. More specifically, if two nodes are connected by an edge, the corresponding variables interact probabilistically in a way that is not mediated by any of the other variables. This allows the network to encode a set of conditional independence assumptions. In a Markov network, the Markov blanket for a node consists of all of its neighbors in the graph [8]. A node is conditionally independent of all other nodes in the graph given its Markov blanket. As a result, the Markov blanket of a node can be used to determine the probabilities of the node variable's possible instantiations without needing to know the states of every other variable in the network.

The Markov network framework captures affinities between related variables by modeling their local interactions via a factor ϕ, which is a function that maps a set of random variables to $\mathbb{R}_{\geq 0}$. Using these functions, the joint probability

distribution can be factorized over the cliques $\mathbf{C}$ of the network's graph structure as

$$P(\mathbf{X}) = \frac{1}{Z} \prod_{c \in \mathbf{C}} \phi(c)$$

where Z is a normalizing constant. This approach provides a great degree of flexibility when representing the interactions between variables. The strength of the interaction can be adjusted by modifying ϕ. A Markov network's structure can be learned from data, or a known structure can be provided by a domain expert.

2) Inference: Exact inference in probabilistic graphical models is known to be NP-Hard [7]. Thus, for practical applications, it is necessary to use an approximation algorithm to generate samples. For an undirected model such as a Markov network, a popular choice for this task is Gibbs sampling. This sampling approach belongs to a category of algorithms known as Markov chain Monte Carlo methods, which generate a sequence of samples generated from distributions that approximate the desired distribution better and better as the sequence continues.

Gibbs sampling starts by generating an initial sample based on an initial distribution, often uniform. Then, for a specified number of iterations, the sampler re-samples each variable based on the current sampled values for the other variables in the network. Markov networks have a property known as the local Markov property, which states that a variable is conditionally independent of all other variables given its neighbors in the graph. Thus, when re-sampling a variable X, it is only necessary to calculate $P(X|Neighbors(X))$, which can be done using the current sampled values for $Neighbors(X)$. For each possible value $x_i \in Vals(X)$, the likelihood of X taking on value x_i is calculated using the values of the potential function associated with the instantiation where $X = x_i$ and $Neighbors(x)$ have the values from the current sample. These likelihoods are then normalized to create the distribution from which X can be resampled.

C. Estimation of Distribution Algorithms

1) Introduction to EDAs: Estimation of Distribution Algorithms are somewhat semantically related to ICPSO in that they are based on the idea of building a probability distribution to reflect desirable candidate solutions [2]. The general EDA procedure begins by randomly generating an initial population of solutions. Some subset of these solutions are selected based on their fitness, similar to the selection procedure in a genetic algorithm. Then, the algorithm constructs a probabilistic model using these selected solutions. In some cases, this includes structure as well as parameter learning; in others, the structure is known beforehand and fixed. This model is then sampled to create more candidate solutions, which replace some portion of the previous generation's population, and the process repeats. Over multiple iterations, this should lead to a population with increasing average fitness.

Often the probabilistic model used in an EDA is a Bayesian network. However, Bayesian networks are restricted to be directed acyclic graphs. In the case of a constrained optimization problem, this can be a restriction on the representational power of the model as constraints can be bidirectional and cyclic in nature. Thus, we focus our review of literature on the class of discrete EDAs that use an undirected model, a Markov network, which permits cycles. Additionally, we are interested primarily in EDAs that can represent higher-order interactions between variables, rather than univariate or bivariate EDAs. Univariate models assume problem variables are independent, and bivariate models similarly restrict the allowable number of interdependencies between variables, limiting the representational power of the model. Given that our algorithm does not place a restriction on the number of interdependencies in the model, we chose to compare to EDAs that had similar representational power.

2) Markov Network Factored Distribution Algorithm (MN-FDA): MN-FDA is an EDA that uses a junction graph as the underlying probabilistic model [9]. The algorithm uses independence tests to learn a Markov network structure from the population. The network structure can also be given as an input, rather than being learned by the algorithm. This network is then modified by incrementally deleting edges between nodes that have degree larger than some specified maximum, in order to decrease the density of the graph. This refinement is necessary to manage the size of the resulting cliques; however, it also has the potential to remove dependencies from the graph beyond what is necessary. The algorithm then uses this network to construct a junction graph from which additional points are sampled.

While earlier EDAs based on undirected models generally used a junction tree to represent factorizations, the use of a junction graph allows cycles and thus extends the representational power of the model. This means that the junction graph can more closely fit the underlying distribution [10].

3) Markov Network Estimation of Distribution Algorithm (MN-EDA): Santana proposed MN-EDA as a novel method to approximate and sample probability distributions [11]. The learning portion of MN-EDA is accomplished through the use of a modified version of the Kikuchi approximation of energy from statistical physics, and the sampling portion of the algorithm relies on Gibbs sampling. The Kikuchi approximation was chosen because it has a number of properties that are particularly useful in an EDA context. When compared to MN-FDA and its reliance on junction graphs, MN-EDA's use of the Kikuchi approximation covers a higher number of dependencies, which can result in accuracy gains. Additionally, these dependencies can be captured without adding new edges to the underlying graph, making this a preferable choice to procedures like triangulation algorithms.

Although MN-EDA and MN-FDA are related in that they both use Markov networks, there are important differences between the two methods, both in terms of approach and performance. A primary difference between the two is that MN-FDA is incapable of approximating probability distributions using "messy factorizations." The MN-FDA approach requires an ordered factorization, one in which there exists

an ordering of all maximal cliques such that at least one variable in each clique does not belong to any of the prior cliques in the ordering. MN-EDA has the ability to learn messy factorizations, and this fact, combined with the advantages gained through the use of the Kikuchi approximation, results in a more robust EDA.

The evaluation of the MN-EDA approach shows that the Kikuchi approximation is capable of encoding information about the problem's structure, and unlike some of the competing EDA approaches, they have no problems handling variables whose interactions form cycles. This makes MN-EDA, Santana contends, a viable and attractive alternative to Bayesian network based EDAs.

4) Distribution Estimation Using Markov Networks (DEUM): DEUM is another class of Markov-network based EDAs; however, what distinguishes DEUM from similar approaches is its use of *fitness modeling* to estimate the distribution [12]. Fitness modeling is a process that can potentially improve the efficiency of evolutionary algorithms by using a model of the fitness function, rather than the function itself [13]. In situations where the fitness evaluation is the primary bottleneck, the increased speed of evaluation by sampling the model is worth the cost of building an explicit model of fitness. DEUM, in particular, relies on the Markov random field fitness model (MFM).

The DEUM algorithm proceeds much like other EDA algorithms, with an added step: when building the probabilistic model, DEUM first builds an MFM and then proceeds to fit it to the set of selected solutions. In the full DEUM algorithm, this step is also preceded by estimation of the structure of the Markov network; however, DEUM can also take advantage of preexisting knowledge of network structure. The authors found that DEUM requires a very large population size to converge to the global optimum when performing structure learning. Like many other multivariate Markov-network-based EDAs, DEUM uses Monte Carlo methods, specifically Gibbs sampling, in order to sample from the model at each iteration.

5) Markovianity Based Optimization Algorithm (MOA): MOA is unusual among EDAs in that it uses the local rather than global Markov property in estimating and sampling from the model [14]. In other words, rather than having to estimate the parameters of the full joint distribution, the algorithm needs only to estimate the conditional probabilities for each node and its Markov blanket and sample from those. Because of this, the algorithm does not need to rely on methods such as Kukuchi approximations, and thus gives a more efficient optimization.

Like DEUM and similar methods, MOA uses Gibbs sampling to sample from the Markov network. Specifically, a linear temperature schedule is used for the sampling process, where at each iteration the temperature is $T = \frac{1}{iteration * CR}$. This temperature parameter controls the convergence of the sampling process, where a higher temperature pushes the distribution closer to being uniform. The CR parameter is a user-defined "cooling rate" between 0 and 1. By using this cooling rate parameter in conjunction with the iteration to determine the temperature, the convergence of the MOA algorithm may be slowed down or sped up. The authors compared this version of Gibbs sampling to temperature-less scheduling, and found that this version was preferable for MOA due to its handling of the exploration/exploitation tradeoff.

Algorithm 1 MICPSO

```
1: INITIALIZESWARM()
2: repeat
3:     for all p ∈ Swarm do
4:         v = ωv + φ1r1 ⊗ (pB − p) + φ2r2 ⊗ (gB − p)
5:         p ← p + v
6:         p.EVALUATEFITNESS()
7:         if f(pBest_sample) > f(gBest_sample) then
8:             gBest_sample ← pBest_sample
9:             gBest ← BIAS(p)
10:        end if
11:    end for
12: until stopping criterion is met return gBest_sample
```

Algorithm 2 EvaluateFitness

```
1: for i = 1 to numSamples do
2:     s ← GIBBSSAMPLE(p.MN)
3:     if f(s) > f(pBest_sample) then
4:         pBest_sample ← s
5:         pBest ← BIAS(p)
6:     end if
7:     fitness ← fitness + f(s)
8: end for
9: fitness ← fitness / numSamples return fitness
```

III. MICPSO

Our approach, Markovian ICPSO (MICPSO), extends ICPSO by using a Markov network, rather than a vector of distributions, to represent the particle's position. This maintains the probabilistic interpretation of the particle's position, but also allows explicit representation of dependency relationships between solution variables. Algorithm 1 shows the basic procedure of MICPSO, while Algorithm 2 provides additional detail on the fitness evaluation procedure.

The algorithm begins, as do many swarm-based algorithms, by creating a randomly-initialized population of solutions. If the structure of the Markov network for the optimization problem is known, this only involves randomly generating the potentials on the edges of the network. Otherwise, however, this must also include a structure learning step of some kind. Then, the algorithm iterates through all particles in the swarm, updating their position and velocity at each iteration. The position and velocity update equations are identical to those in regular ICPSO, except that they apply to the potentials in the network, rather than entries in a probability distribution. Note that $r_1, r_2 \sim U(0, 1)$. As in ICPSO, the fitness of a particle is assessed by generating samples; however, in this case, this requires performing inference on the particle's Markov network. We use Gibbs sampling for this step. The fitness of the generated samples is averaged to obtain the particle's

fitness. If at any point during sampling a particle is produced that beats the global or local best, the best is updated and the particle's position vector is biased toward producing similar samples in the future. MICPSO uses the same biasing method as ICPSO, adjusting the position vector by increasing entries corresponding to the best sample's solution values and decreasing the other entries. In MICPSO, the resulting distribution does not need to be re-normalized after the biasing step, since it does not encode an explicit probability distribution. Once the algorithm terminates, the best sample generated by the global best particle is returned as the solution.

The use of the probabilistic graphical model is the reason our review of related literature focuses primarily on EDAs, as we view this as playing a similar role to the Markov network in our algorithm. Both our method and EDAs rely on building a model of possible high-fitness solutions; however, the means by which this is achieved differs significantly between EDAs and MICPSO. An EDA builds this model from a population, while our method centers on adjusting the parameters of the Markov network directly in response to sample fitness.

IV. Experiments

Testing the validity and effectiveness of the proposed MICPSO approach necessitated selecting a categorical optimization problem where the solution values are not independent of each other. To that end, we chose to use the simplest form of the graph coloring problem, vertex coloring. For a vertex coloring problem, the goal is to color a graph G such that no two vertices that share an edge are the same color. The minimum number of colors required so that no adjacent nodes in G share a color is known as the chromatic number of the graph, often denoted $\chi(G)$. Finding the chromatic number of a graph is NP-hard, so our goal in testing algorithms on this problem is to see how closely they can approximate the chromatic number of various graphs [15]. This provides us with a simple and intuitive method of scoring the algorithms.

A. Methodology

The vertex coloring problems used in these experiments were drawn from [16]. Instances were chosen to represent a range of graph sizes and complexities, as reflected by the number of nodes and/or edges in the graphs, as well as treewidth. As a result, these problems varied considerably in difficulty. Table I lists the characteristics of each of the graphs used, including the number of nodes and edges, the treewidth, and the chromatic number of the graph. The treewidth corresponds to the size of the largest subclique in the graph (minus 1) after triangulating and gives an idea of the complexity of each problem: higher treewidth indicates a more difficult coloring problem. All graphs are non-planar, and all tree widths are drawn from the results of [17], which demonstrated a Branch and Bound algorithm called $QuickBB$ for calculating upper bounds on treewidth.

We chose to use the vertex coloring problem when testing the performance of MICPSO because it is a well-researched problem that extends to a number of real-world applications.

TABLE I
Characteristics of each of the DIMACS graph coloring instances tested in this work.

	Vertices	Edges	TW	$\chi(G)$
huck	74	301	10	11
myciel3	11	20	5	4
myciel4	23	71	10	5
myciel5	47	236	19	6
queen5-5	25	160	18	5

A valid graph coloring can be employed to represent solutions to various versions of the scheduling problem, optimal register allocation, and minimizing the number of frequencies required to assign antennas in a wireless network [18], [19]. Additionally, examples of graph coloring problems are plentiful and solutions are easy to score. Importantly, this problem also allows the algorithm to take advantage of known network structure rather than incorporating a structure learning step.

Along with MICPSO and ICPSO, we also chose to test IPSO and MOA on these problems. MOA was selected because, of the Markov network-based EDAs surveyed, it was the most semantically similar to MICPSO and thus we felt it would provide the best comparison. Additionally, we implemented a simple hillclimber to serve as a baseline for comparison. Neither VPSO nor PPSO were included in this comparison, as it has already been shown that they are generally outperformed by ICPSO on categorical problems [1].

Each algorithm was run 20 times on each graph instance, and results were averaged across all runs. Each algorithm's parameters were tuned individually prior to the comparison. Both MICPSO and ICPSO performed best when the swarm size was 10 and 3 samples were generated per fitness evaluation. MICPSO performed best with $\epsilon = 0.75$, $\omega = 0.7$, and $\phi_1 = \phi_2 = 1.4$. ICPSO used $\epsilon = 0.95$, $\omega = 0.7$ and $\phi_1 = \phi_2 = 1.4$. IPSO's parameters were $\omega = 1.0$, and $\phi_1 = \phi_2 = 1.4$, with a swarm size of 10. Finally, MOA performed best with a population of size 25, a cooling rate of 0.1, and 50% of the population replaced at each iteration. As in [14], we used truncation selection for MOA. However, we did not incorporate MOA's structure learning step, instead allowing both MOA and MICPSO to make use of the known network structure for graph coloring problems.

Algorithms were run until the best solution's fitness failed to change over 20 consecutive iterations, at which point the global best was returned. We chose this stopping criterion, rather than a number of fitness evaluations, based on the results from [20]. Given that this is a constrained optimization problem, we had to choose a way to handle the constraints in the methods that did not incorporate them explicitly. Broadly speaking, there are three ways this can be done. When an invalid solution is generated, the algorithm can either (1) fix the solution to make it valid, (2) discard that solution and generate a new one, repeating until a valid solution is produced, or (3) penalize the fitness of the invalid solution so that, ideally, it will not be chosen as a best solution [21].

With a problem as tightly constrained as graph coloring, the first two options could potentially be prohibitively computationally expensive. Employing an auxiliary algorithm to repair an invalid solution could take a long time on the larger graphs, and with highly connected graphs, it might be necessary to generate thousands of solutions before a valid one is produced. Furthermore, it is possible for ICPSO to produce a distribution that is incapable of generating a valid sample at all; therefore, the second option could potentially create an infinite loop. For these reasons, we chose to utilize a penalty function to reduce the fitness of invalid solutions. This penalty, however, was not factored into the reported results for average fitness per iteration or average fitness of the global best solution, as this would have unfairly biased the results. With this penalty in place, our fitness function for a vertex coloring C of graph G is as follows:

$$f(C(G)) = \begin{cases} \chi(G) - |C| & \text{if } C \text{ is a valid coloring} \\ (\chi(G) - |C|) - 100 & \text{otherwise.} \end{cases}$$

MOA presented an interesting problem in terms of constraint enforcement because, while it is Markov network-based like MICPSO, the conditional probabilities of variables' values are computed from the population rather than the network's potential functions. At each iteration of the MOA Gibbs sampler, the probability that variable X_i takes on value x_i is computed as follows:

$$p(x_i|N_i) = \frac{e^{p(x_i,N_i)/T}}{\sum_{x_i'} e^{p(x_i',N_i)/T}}$$

where T is the temperature variable and $p(x_i, N_i)$ and $p(x_i', N_i)$ are calculated from the population [14]. Due to the use of the Boltzmann distribution, the numerator can never go to zero; therefore, it is not possible to have a zero probability even in the case where $X_i = x_i$ might violate a graph coloring constraint given the assignment to the neighbors N_i. As such, it was still possible to generate an invalid sample using MOA. In order to mitigate the effects of this and allow for a more fair comparison to MICPSO, we modified MOA to discard invalid samples during the sampling process, re-sampling until an adequate number of valid samples have been produced.

B. Results

The results of these experiments are shown in Table II. Results were compared using Welch Two Sample t-tests. The "NR" in the table indicates that the algorithm was unable to produce any valid results on that graph, so no fitness values may be calculated. "N/A" indicates that the algorithm ran out of memory before producing any results on the graph. An entry in the table is bolded if that algorithm statistically significantly outperformed all other algorithms on that problem.

Table III reports the percentage of the vertex colorings produced by each algorithm that were valid for each graph. Again, an "N/A" entry indicates that no results were obtainable in that case. The fitness curves for myciel3 and myciel4 – the only graphs where fitness information was available for every method – are shown in Figures 1a and 1b.

TABLE II
MEAN FITNESS VALUES FOR EACH OF THE GRAPHS TESTED.

	HC	IPSO	ICPSO	MICPSO	MOA
huck	-25.57	-32.43	-34.35	**-0.24**	N/A
myciel3	-4.50	-3.30	-3.95	**0.00**	-3.20
myciel4	-7.67	-10.8	-11.50	**-0.80**	-8.35
myciel5	-15.75	-26.14	-26.45	**-2.40**	N/A
queen5-5	NR	-15.40	-14.90	**-4.00**	N/A

TABLE III
PERCENTAGE OF THE TIME EACH METHOD WAS ABLE TO FIND A VALID SOLUTION FOR EACH GRAPH TESTED.

	HC	IPSO	ICPSO	MICPSO	MOA
huck	35	35	100	100	N/A
myciel3	10	100	100	100	100
myciel4	15	100	100	100	100
myciel5	40	70	100	100	N/A
queen5-5	0	100	100	100	N/A

C. Analysis

MICPSO statistically significantly outperformed all other algorithms on all problems tested. Especially as compared to ICPSO, this reinforces the notion that MICPSO's ability to represent dependencies directly gives it an advantage on problems where solution variables are dependent. Additionally, while one might have expected ICPSO to outperform IPSO given the categorical nature of the problem, their performance was generally comparable. This suggests that, for vertex coloring, the constraints and dependencies matter more to solution quality than the fact that it is a categorical problem.

MOA was only able to produce results for the two smallest graphs, myciel3 and myciel4. It performed second-best on myciel3, but actually came in third to the baseline hillclimber on myciel4. However, it is worth noting that the hillclimber was only able to produce a valid result in a small percentage of cases, while the vertex colorings produced by MOA were always valid. Despite also incorporating dependencies using a Markov network, MOA was consistently outperformed by MICPSO. We hypothesize that this is due in part to MICPSO's ability to incorporate the constraints of the problem explicitly.

Overall, all algorithms had relatively more difficulty with graphs of larger treewidth. This is unsurprising, as treewidth is an indicator of the complexity of the graph. Notably, myciel5 and queen5-5 had the largest treewidths of the graphs tested, and MICPSO in particular had the most difficulty with those two graphs. Huck also had the lowest average fitness for all algorithms but MICPSO, suggesting the other methods also had difficulty with large graphs even with a moderate treewidth. Interestingly, on some problems, the hillclimb outperformed IPSO and ICPSO in terms of mean solution fitness. As previously mentioned, however, it was unable to find a valid solution in the majority of cases.

MOA and MICPSO inherently always produce a valid sample, when a sample is able to be produced at all. IPSO, ICPSO, and the hillclimber were not constrained to always

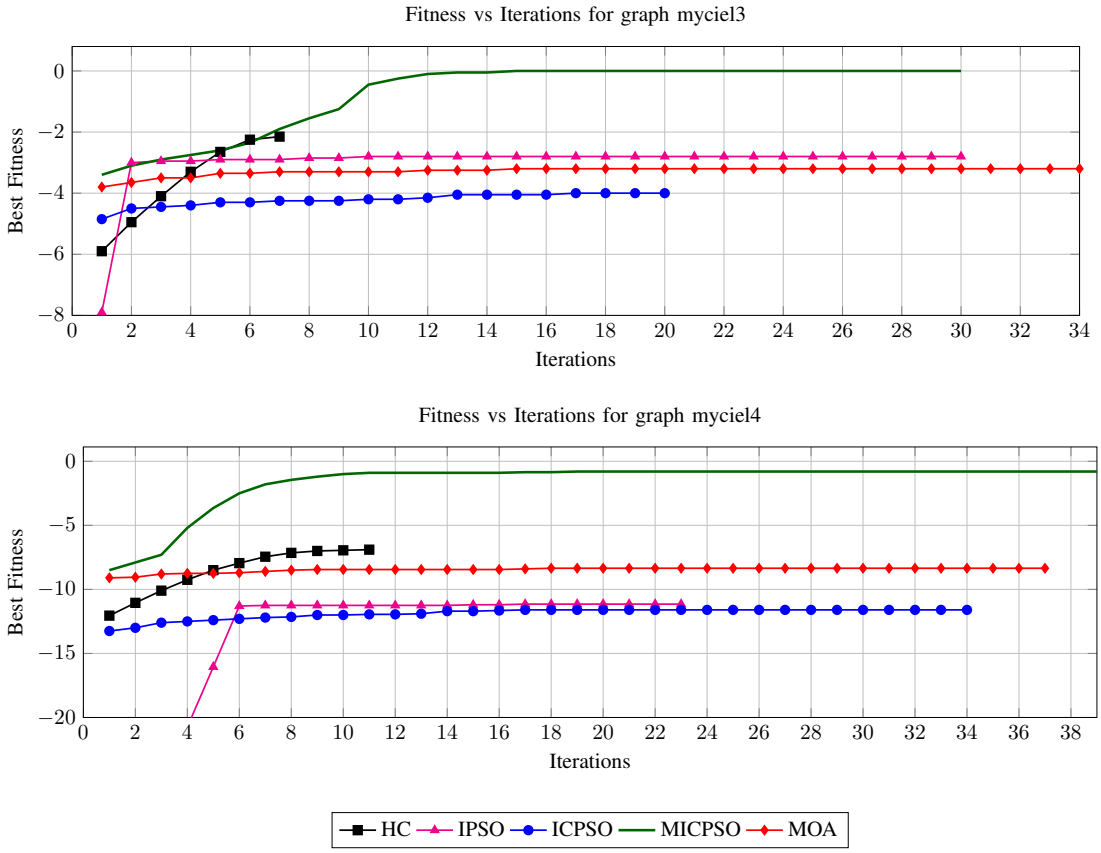

Fig. 1. Fitness curves for optimizing the number of colors used for graphs myciel3 and myciel4

produce a valid sample, and thus often a penalized solution was returned as the best result. ICPSO was able to produce valid solutions consistently on the graphs presented, but failed on some even larger and more complex graphs during the early stages of testing. Along with the hillclimber, IPSO had significant difficulty producing valid solutions, particularly on networks with a relatively large number of edges.

Generally speaking, the Markov network-based approaches were the slowest to produce results. MOA in particular was computationally expensive compared to the other methods, as evidenced by the fact that it often failed altogether to find a solution before running out of memory. The use of Gibbs sampling also caused MICPSO to be slower than the simpler methods; however, even on the larger graphs, the computational time required was not so slow as to be unusable, and the algorithm always returned a solution on all five graphs.

Analysis of the fitness curves shows that MICPSO consistently produces the fittest solutions, and it does so within a relatively small number of iterations. In fact, on average, even the solutions produced by MICPSO in the first few iterations are better than the other methods' solutions. MICPSO generally reached decreasing returns within 12 iterations, suggesting that our termination criterion was possibly more conservative than necessary. Given the computational cost incurred by Gibbs sampling, it seems desirable in light of these results to give MICPSO a tighter termination criterion, as that would lead to fewer iterations overall while still preserving solution quality. The other algorithms tested tended to reach decreasing returns even sooner than MICPSO, but on average the solutions reached had lower fitness.

IPSO had an extremely steep initial fitness gain, but then essentially leveled out within a single-digit number of iterations. The hillclimber had similar results, but with a less dramatic initial increase. On the other hand, both ICPSO and MOA tended to remain fairly stagnant in terms of best solution fitness. In MOA's case, this suggests an inability to introduce or maintain diversity in the population throughout the course of optimization. We hypothesize this could be due to MOA's use of truncation selection, which can cause a loss of diversity compared to other selection methods [22]. However, it could also be due to the fact that the parameters of the Markov network are learned from the population. While the use of the Boltzmann distribution may help to mitigate this effect, it is still likely that the algorithm will be unable to escape local optima due to its reliance on the distribution of the population at the previous step.

V. Conclusion

In this paper, we introduced an extension to ICPSO, MICPSO, which utilizes a Markov network to represent the particle's position. By doing so, a particle can represent

dependencies between state variables explicitly and incorporate some types of constraints. This allows the algorithm to avoid fixing, discarding or penalizing infeasible solutions, and also is shown to increase solution quality. We compared MICPSO to ICPSO, IPSO, MOA, and a hillclimber on a set of benchmark vertex coloring problems and found that MICPSO consistently outperformed the other algorithms. This suggests that MICPSO successfully incorporates dependencies into the underlying discrete PSO in a way that is applicable to a well-known problem from the discrete optimization literature.

In future work, it will be important to test MICPSO's performance on a variety of other problems. The results on vertex coloring are promising, but that represents only one of many potential benchmark discrete problems to which it could be applied. Additionally, while vertex coloring presents an interesting first application, the presence of constraints in the optimization problem limits its use in this context beyond basic proof of concept. While MICPSO can explicitly incorporate constraints of the type present in vertex coloring by simply clamping certain values of the potential functions to zero, we expect that a traditional constraint solver would still outperform MICPSO on this problem.

A second, related area of future work would be incorporating a structure learning step into MICPSO. While we already demonstrated that the algorithm can be effective for applications where the Markov network has a known structure, it will be important to test the algorithm's performance when the structure must be learned from data, as is the case in many applications of discrete optimization. Even disregarding real-world problems, many classic theoretical computer science problems, for example, the job shop scheduling problem and activity selection, lack an immediately obvious Markov network structure. Learning the network structure for a target problem will allow MICPSO to move away from applications like vertex coloring, which are better suited to constraint solver algorithms, and into areas where evolutionary and population-based approaches can be expected to outshine other methods.

Another possibility would be developing a factored version of MICPSO. Factored evolutionary algorithms (FEA) have already been shown to be a promising area of research for evolutionary algorithms, including swarm-based methods [23]. The inclusion of the Markov network in MICPSO particles may make the algorithm a strong candidate for a factored approach to optimization.

Finally, it would be interesting to compare MICPSO to other optimization methods that had been modified to better represent underlying joint probability distributions. As demonstrated by MOA, using a Markov network to encode dependency information is not limited to particle-based methods, and could potentially extend to other probability-based methods.

Acknowledgments

We would like to thank the members of the Numerical Intelligent Systems Laboratory (NISL) at Montana State University for their input during the development stages of this paper.

References

[1] S. Strasser, R. Goodman, S. Butcher, and J. W. Sheppard, "A new discrete particle swarm optimization algorithm," in *Proceedings of the Genetic and Evolutionary Computation Conference*, 2016, p. To appear.

[2] M. Hauschild and M. Pelikan, "An introduction and survey of estimation of distribution algorithms," *Swarm and Evolutionary Computation*, vol. 1, no. 3, pp. 111–128, 2011.

[3] J. Kennedy and R. Eberhart, "Particle swarm optimization," in *Proceedings of the IEEE International Conference on Neural Networks*, 1995, pp. 1942–1948.

[4] K. Veeramachaneni, L. Osadciw, and G. Kamath, "Probabilistically driven particle swarms for optimization of multi valued discrete problems: design and analysis," in *2007 IEEE Swarm Intelligence Symposium*. IEEE, 2007, pp. 141–149.

[5] J. Pugh and A. Martinoli, "Discrete multi-valued particle swarm optimization," in *Proceedings of IEEE swarm intelligence symposium*, no. SWIS-CONF-2006-004, 2006, pp. 103–110.

[6] B. Taskar, V. Chatalbashev, and D. Koller, "Learning associative Markov networks," in *Proceedings of the Twenty-first International Conference on Machine Learning*, ser. ICML '04. New York, NY, USA: ACM, 2004.

[7] D. Koller and N. Friedman, *Probabilistic graphical models: principles and techniques*. MIT press, 2009.

[8] J. Pearl, *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1988.

[9] R. Santana, "A Markov network based factorized distribution algorithm for optimization," in *Machine Learning: ECML 2003*. Berlin Heidelberg: Springer, 2003, pp. 337–348.

[10] A. E. Brownlee, "Multivariate Markov networks for fitness modelling in an estimation of distribution algorithm," *Open Access Institutional Repository at Robert Gordon University*, 2009.

[11] R. Santana, "Estimation of distribution algorithms with Kikuchi approximations," *Evolutionary Computation*, vol. 13, no. 1, pp. 67–97, Jan. 2005.

[12] S. Shakya, A. E. Brownlee, J. McCall, F. Fournier, and G. Owusu, "A fully multivariate DEUM algorithm," in *Evolutionary Computation, 2009. CEC'09. IEEE Congress on*. IEEE, 2009, pp. 479–486.

[13] A. E. Brownlee, J. A. McCall, and Q. Zhang, "Fitness modeling with Markov networks," *IEEE Transactions on Evolutionary Computation*, vol. 17, no. 6, pp. 862–879, 2013.

[14] S. Shakya, R. Santana, and J. A. Lozano, "A Markovianity based optimisation algorithm," *Genetic Programming and Evolvable Machines*, vol. 13, no. 2, pp. 159–195, 2012.

[15] M. R. Garey, D. S. Johnson, and L. Stockmeyer, "Some simplified NP-complete problems," in *Proceedings of the Sixth Annual ACM Symposium on Theory of Computing*, ser. STOC '74. New York, NY, USA: ACM, 1974, pp. 47–63.

[16] "Graph coloring instances," http://mat.gsia.cmu.edu/COLOR/instances.html, accessed: 2016-06-30.

[17] V. Gogate and R. Dechter, "A complete anytime algorithm for treewidth," in *Proceedings of the 20th conference on Uncertainty in artificial intelligence*. AUAI Press, 2004, pp. 201–208.

[18] K. I. Aardal, S. P. M. Hoesel, A. M. C. A. Koster, C. Mannino, and A. Sassano, "Models and solution techniques for frequency assignment problems," *Annals of Operations Research*, vol. 153, no. 1, pp. 79–129, 2007.

[19] G. J. Chaitin, "Register allocation & spilling via graph coloring," in *Proceedings of the 1982 SIGPLAN Symposium on Compiler Construction*, ser. SIGPLAN '82. New York, NY, USA: ACM, 1982, pp. 98–105.

[20] A. P. Engelbrecht, "Fitness function evaluations: A fair stopping condition?" in *Swarm Intelligence (SIS), 2014 IEEE Symposium on*. IEEE, 2014, pp. 1–8.

[21] Q. He and L. Wang, "An effective co-evolutionary particle swarm optimization for constrained engineering design problems," *Engineering Applications of Artificial Intelligence*, vol. 20, no. 1, pp. 89–99, Feb. 2007.

[22] T. Blickle and L. Thiele, "A comparison of selection schemes used in evolutionary algorithms," *Evolutionary Computation*, vol. 4, no. 4, pp. 361–394, 1996.

[23] S. Strasser, J. Sheppard, N. Fortier, and R. Goodman, "Factored evolutionary algorithms," in *IEEE Transactions on Evolutionary Computation*, 2016, p. Under review.