Evolving Mixtures of n-gram Models for Sequencing and Schedule Optimization

Chung-Yao Chuang and Stephen F. Smith

The Robotics Institute, Carnegie Mellon University
5000 Forbes Avenue, Pittsburgh, PA 15213, USA
cychuang@cmu.edu, sfs@cs.cmu.edu

Abstract. In this paper, we describe our work on Estimation of Distribution Algorithms (EDAs) that address *sequencing problems*, i.e., the task of finding the best ordering of a set of items or an optimal schedule to perform a given set of operations. Specifically, we focus on the use of probabilistic models that are based on n-gram statistics. These models have been used extensively in modeling statistical properties of sequences. We start with an EDA that uses a bigram model, then extend this scheme to higher-order models. However, directly replacing the bigram model with a higher-order model often results in premature convergence. We give an explanation on why this is the case along with some empirical support for our intuition. Following that, we propose a technique that combines multiple models of different orders, which allows for smooth transition from lower-order models to higher-order ones. Furthermore, this technique can also be used to incorporate other heuristics and prior knowledge about the problem into the search mechanism.

1 Introduction

Estimation of Distribution Algorithms (EDAs) are a class of population-based stochastic search techniques that search the solution space by learning and sampling probabilistic models [1,2]. Using probabilistic models in the search mechanism enables EDAs to adopt techniques from machine learning and statistics to automatically discover patterns exhibited in a set of promising solutions. In past studies, EDAs have been applied to a variety of academic and real-world optimization problems [2,1], achieving competitive results in many scenarios: chemical applications [3], power systems [4], and environmental resources [5], to name a few. Most of these studies were focused on domains in which a solution can be naturally represented as a fixed-length string with no ordering dependencies. However, many interesting and important optimization problems require the determination of a best ordering of a set of items or an optimal sequence to perform a given set of operations. In this work, we are interested in solving such kind of *sequencing problems* through the paradigm of EDAs.

One classical example of this kind of problem is the Traveling Salesman Problem (TSP). The objective of the TSP is to find the shortest route for a traveling salesman who is on the mission to visit every city on a given list precisely once and then return to the initial city. This task is equivalent to finding the Hamiltonian cycle that has the smallest cost in a complete weighted graph. The TSP is

T. Bartz-Beielstein et al. (Eds.): PPSN XIII 2014, LNCS 8672, pp. 312–321, 2014.

celebrated because many scientific and engineering problems can be formulated as TSPs and it has long been used to study sequencing and scheduling problems. In this paper, we will use the TSP as our model problem for evaluation purposes.

Some previous works can be found in the literature that deal with sequencing and scheduling problems by means of EDAs. However, most of these approaches are direct adaptations of EDAs designed for discrete or continuous problems that have no ordering properties. Earliest attempts [6] applied discrete EDAs as if a solution has no sequential dependencies. The obvious drawback is that the information of relative ordering among items is not explicitly considered in the constructed models. This deficiency may be the cause of low success rate in finding the global optimum as reported in [6,7,8]. Adaptation of continuous EDAs [6,9] has also been explored. Most of the research in this direction uses the random keys representation [10]. Using this representation, some of the information about the relative ordering of the items can be encoded in the probabilistic model. However, with this type of construct, an algorithm has to search for solutions in a largely redundant real-valued space. This inefficiency is reflected in their relatively inferior performance in the review by Ceberio et al. [11].

The limitation of these direct adaptation of EDAs designed for problems without ordering properties encourage the EDA community to invent other approaches that specifically target the sequencing problems. More relevant to this research is the work done by Tsutsui et al. [7,8]. They proposed an approach called Edge Histogram Based Sampling Algorithm (EHBSA), which constructs an edge histogram matrix by counting the number of occurrences that item i and item j appear consecutively in the sequences. For TSP, this is how many times the link between the i-th and j-th city is observed in promising solutions. Based on this statistics, a distribution is estimated that gives conditional probability of the next item given the previous one. This approach is equivalent to estimating a bigram model from the current population. In this research, we work on generalizing this idea to n-gram models.

Although this generalization seems straightforward, as we will show empirically, the naive approach of increasing the order of the model (e.g., using trigram instead of bigram) does not work. Instead, we developed a method that uses linear interpolation to combine multiple models of different orders, and utilize a holdout set to estimate the weight associated with each model. In this way, we can gradually shift the emphasis from a low-order model to higher order ones as longer patterns emerge in the population. Furthermore, as we will show in Sect. 5, this technique can also be used to incorporate other heuristics and prior knowledge about the problem into the search mechanism.

In the next section, we will describe the formulation of the n-gram models. After that, Sect. 3 introduces our approach of using n-gram models for guiding the search process. It also discusses the difficulty encountered when moving from bigram model to higher-order models. In Sect. 4, we present a method that is able to combine multiple models of different orders, and thus provides a smooth transition from lower-order model to higher-order ones. Sect. 5 further describes how we can use this same method to incorporate other heuristics and prior knowledge about the problem into the search mechanism. We briefly discuss some characteristics of our proposal in Sect. 6. Finally, Sect. 7 summarizes this paper and points out the future direction of our work.

2 Modeling Sequence Properties with *n*-gram Statistics

An n-gram is a pattern of n consecutive items, which is usually a segment from a longer sequence. Such a construct is often used in the tasks of modeling statistical properties of sequences, especially in the field of natural language processing (NLP). For example, a classic task in NLP is to predict the next word given the previous words. Such task can be stated as attempting to estimate the conditional probability of observing some item w_i as the next item given the history of items seen so far. The n-gram approach to this estimate is to make a Markov assumption that only prior local context—the last few items—affects the next item. More formally, we are interested in estimating

$$P(W_i = w_i | W_{i-n+1} = w_{i-n+1}, \ldots, W_{i-1} = w_{i-1})$$

where the sequence $w_1, w_2, \cdots$ is some instantiation of a sequence of random variables $W_1, W_2 \cdots$. In the following, we will use $P(w_i | w_{i-n+1} \cdots w_{i-1})$ as a shorthand for this probability function.

The obvious first answer to the above formulation is to suggest using a *maximum likelihood estimate* (MLE):

$$P_{\text{MLE}}(w_i | w_{i-n+1} \cdots w_{i-1}) = \frac{C(w_{i-n+1} \cdots w_{i-1} w_i)}{\sum_{v \in \mathbf{V}} C(w_{i-n+1} \cdots w_{i-1} v)}$$

where $C(w_{i-n+1} \cdots w_{i-1} w_i)$ is the frequency of a certain n-gram in the training samples, and $\mathbf{V}$ is the set of possible items. However, a drawback is that MLE assigns a zero probability to unseen events, which effectively zeros out the probability of sequences with component n-grams that just happened not appearing in the training samples. For our scenario, this creates a risk of arbitrarily discarding some portion of the unexplored search space. Thus, we need a more suitable estimator that takes previously unseen patterns into consideration.

A simple solution to this problem is to smooth the distribution with some *pseudocount* κ:

$$P_\kappa(w_i | w_{i-n+1} \cdots w_{i-1}) = \frac{C(w_{i-n+1} \cdots w_{i-1} w_i) + \kappa}{\sum_{v \in \mathbf{V}} (C(w_{i-n+1} \cdots w_{i-1} v) + \kappa)}$$

where κ is usually set to a value smaller than 1. In this work, we use this simple method to allocate probability mass for unobserved events, though more sophisticated estimators are possible for this task.

3 An EDA Framework with *n*-gram Models

This section describes the basic approach that uses n-gram models in the EDA framework for sequencing and scheduling problems. To briefly recap the operations performed by an EDA: At each iteration, we start with a set of promising solutions, then the algorithm constructs a probabilistic model based on the statistics gathered from those solutions. Once a model is learned, a number of new solutions will be generated by sampling the model to replace solutions in the current population according to some replacement strategy.

Table 1. Observations on solving `gr48`

Method	Success Rate	# of Evaluations	
		mean	std
2G	30/30	277024	34535.4
3G	9/30	224640	118490.2
2G $\xrightarrow{800 \text{ iter.}}$ 3G	30/30	240032	20753.3

In this work, instead of generating an entire solution anew, we first take an existing solution from the current population and randomly extract a subsequence from that solution. This segment will then be taken as the first part of the new solution and serve as the "history" on which the further sampling is based. This kind of *partial sampling* technique has been used by previous researchers such as Chuang and Chen [12] and Tsutsui et al. [8], achieving better usage of diversity and resulting in significant improvement in performance. For our purpose, this has an additional benefit of providing a convenient basis to initialize the sampling from the n-gram models.

Once we have the first part of the new solution, we will further generate the rest of the solution by repeatedly sampling the n-gram model, with previous $n-1$ items as the history. In order to have a valid solution, the set of possible items $\mathbf{V}$ may be varied as the sampling goes on. For example, when dealing with the TSPs, the set of possible next cities $\mathbf{V}$ has to be altered to exclude cities that have already been included in the constructed partial solution.

To summarize the overall flow of the algorithm: At each iteration, we slide a window of size n through each solution in the current population to obtain the frequency counts of n-gram patterns. This statistics is then used for estimating an n-gram model in the form of a conditional distribution. To generate a new solution, we use partial sampling on an existing solution in the current population. Each solution in the current population is visited once for such sampling. Following each partial sampling, a replacement competition is hold between the new solution and the solution from which that new solution's starting segment was extracted. Note that we use replacement as the sole means for selecting promising solutions, i.e., better solutions are preserved under the replacement process. This is similar to the evolution strategies [13], in which every solution in the current population is seen as a potentially good solution because they have survived previous replacement competitions.

As a first step, we examine the performance of using a bigram model

$$P_{2\text{G}}(w_i|w_{i-1}) = \frac{C(w_{i-1}w_i) + \kappa}{\sum_{v \in \mathbf{V}} (C(w_{i-1}v) + \kappa)}$$

for solving a 48-city TSP instance, `gr48`, taken from TSPLIB[1] [14]. Let ℓ denote the problem size. In this experiment, the population size N is set to 5ℓ, the pseudocount is set to $\kappa = 0.01$, and the termination criterion is when either the optimal tour is found or when the algorithm reaches 50ℓ iterations. We ran the algorithm 30 times to observe the average performance. The result of the experiment is presented in Table 1. It shows the success rate in finding the optimal tour and the average number of objective function evaluations used by

[1] `http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/`

the algorithm among the successful runs. It can be seen that the bigram approach gives a pretty decent performance. It shows a high success rate in finding the global optimum, and a reasonable usage of function evaluations.

A tempting thought to proceed is to increase the order of the model. For example, instead of using bigram, we could use a trigram model

$$P_{3G}(w_i|w_{i-2}w_{i-1}) = \frac{C(w_{i-2}w_{i-1}w_i) + \kappa}{\sum_{v \in \mathbf{V}} (C(w_{i-2}w_{i-1}v) + \kappa)}$$

for learning the patterns. However, as shown in Table 1, this results in a significant drop in success rate. Our explanation is that at early stage of a run, there are not so many long patterns that are of good quality. If we attempt to use a higher-order model to learn longer patterns when there are none, we will end up encoding mediocre patterns into the model. Thus, a better way to proceed may be to use a low-order model like bigram model at the beginning of a run and switch to higher-order ones after longer patterns have emerged in the population.

To provide some empirical support for this conjecture, we modified the process to begin with a bigram model, and then switch to the trigram after 800 iterations. As shown in the third row of Table 1, the success rate returns to the same level as using the bigram model and it shows some improvement on function evaluations over the bigram approach. It seems that we can use this technique to gradually move to higher-order models. However, choosing an appropriate schedule to make such switches is a nontrivial task. To address this issue and avoid having to choose a fixed switching point to elevate to a higher-order model, we propose an approach that estimates multiple n-gram models of different orders and combines those models into one composite model. The method automatically calibrates the degree of emphasis placed on each n-gram model. The detail of our formulation is presented in the next section.

4 Combining Multiple Models with Linear Interpolation

Specifically, we formulate the synthesis of multiple models as a linear combination of distributions

$$P(w_i|\mathbf{h}_i) = \sum_j \lambda_j P_j(w_i|\mathbf{h}_i) \tag{1}$$

where $\mathbf{h}_i$ represents the history of items we have seen so far, j is the index to a particular model, and λ_j is the weight associated with the j-th model such that $\lambda_j > 0$ and $\sum_j \lambda_j = 1$. A combination of bigram and trigram model will be

$$P_{2G+3G}(w_i|\mathbf{h}_i) = \lambda_{2G} P_{2G}(w_i|w_{i-1}) + \lambda_{3G} P_{3G}(w_i|w_{i-2}w_{i-1})$$

Assuming that we want to combine K models together, for this formulation, we have to determine K weights, $\lambda_1, \lambda_2, \ldots, \lambda_K$, associated with those models. To do this, we reserve a portion of the population for the task of estimating appropriate values for those λ_j's. Suppose that there are M items in such a holdout set for which we can give conditional probabilities. For each model P_j, we create a probability stream $\mathbf{p}_j = (p_{j1}, p_{j2}, \ldots, p_{jM})$ where p_{ji} is the probability of item w_i predicted by the model P_j, i.e., $p_{ji} = P_j(w_i|\mathbf{h}_i)$. These K probability

Algorithm 1. Estimating the Weights λ_j's

Input: A set of probability streams $\{\mathbf{p}_1, \mathbf{p}_2, \ldots, \mathbf{p}_K\}$, such that each $\mathbf{p}_j = (p_{j1}, p_{j2}, \ldots, p_{jM})$ is of length M.

Initialize $\mathbf{\Lambda}^{(0)} = \{\lambda_1^{(0)}, \lambda_2^{(0)}, \ldots, \lambda_K^{(0)}\}$ s.t. each $\lambda_j^{(0)} > 0$ and $\sum_{j=1}^{K} \lambda_j^{(0)} = 1$
repeat from $t = 0$
  For $j = 1 \ldots K$, update λ_j using $\lambda_j^{(t+1)} = \frac{1}{M} \sum_{i=1}^{M} \frac{\lambda_j^{(t)} p_{ji}}{\sum_{k=1}^{K} \lambda_k^{(t)} p_{ki}}$
  $t = t + 1$
until the difference between $\mathbf{\Lambda}^{(t)}$ and $\mathbf{\Lambda}^{(t-1)}$ is small
return $\mathbf{\Lambda}^{(t)}$

streams (each of length M) are then used as the input to Algorithm 1 [15]. The resulting λ_j's optimize the average likelihood with respect to this holdout set[2].

To illustrate the search behavior, Fig. 1 shows the variation of weights in a typical run that uses a combination of the bigram and trigram models. The weight for the bigram model starts out with a high value and gradually decreases. On the other hand, the weight of the trigram model will begin to dominate in later part of the search, meaning that we do more and more sampling with the trigram model. Based on this self-adaptive behavior, which adjusts the weights automatically, we call our proposal the "evolving mixture."

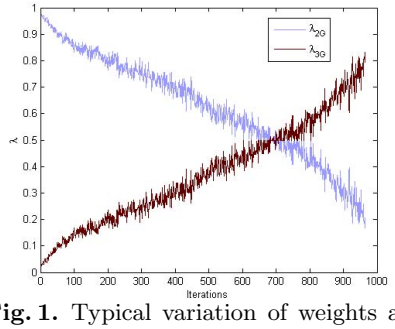

Fig. 1. Typical variation of weights associated with P_{2G} and P_{3G}. This illustration is from a run on `gr48` with $N = 450$ and 30% of the population as holdout set.

To evaluate our proposal, we performed a set of experiments on ten TSP instances from TSPLIB. The parameters to the algorithms are listed in Table 2. As before we ran each algorithm 30 times to give the average performance. The outcomes are presented in Table 3. In each table, we listed the success rate of each algorithm and its average usage of function evaluations among the successful runs. Note that the trailing number in each instance's name represents the number of cities in that instance.

The result of using evolving mixture to combine bigram and trigram is listed in the third row of each table. Comparing to the trigram approach (second row of the table), it gives a significantly better success rate, which is at the same level of the bigram approach. On the other hand, when compared with bigram approach, the evolving mixture approach uses less function evaluations on average.

5 Incorporating Other Heuristics

In its formulation of Eq. (1), we did not put a restriction on the type of the models that can be included. It does not have to be an n-gram model for the

[2] For simplicity, we use $\max_j |\lambda_j^{(t)} - \lambda_j^{(t-1)}| < 0.001$ as condition to terminate Algo. 1.

Table 2. Parameter Settings

(a) For problem size $\ell < 70$

Parameters	Value
population size	$N = 5\ell$
pseudocount	$\kappa = 0.01$
size of holdout set	$R = \lfloor \frac{N}{10} \rfloor$
max iterations	50ℓ

(b) For problem size $\ell >= 70$

Parameters	Value
population size	$N = 5\ell$
pseudocount	$\kappa = 0.001$
size of holdout set	$R = \lfloor \frac{N}{10} \rfloor$
max iterations	80ℓ

evolving mixture to work. The only constraint is that the model takes the form of a (conditional) probability distribution. We can even push the envelop to include something that just *looks like* a probability distribution.

For example, if we want to incorporate a distance-based heuristic for TSP into the search mechanism, we could do so by crafting an "artificial distribution"

$$P_{\text{DH}}(w_i|w_{i-1}) = \frac{d(w_{i-1}, w_i)^{-10}}{\sum_{v \in \mathbf{V}} d(w_{i-1}, v)^{-10}}$$

where $d(u, v)$ is the distance between city u and city v. In short, this formula assigns a larger probability mass to a city that has shorter link to the last city in the partial tour constructed so far.

To see the effect of incorporating such a heuristic, we performed experiments on the same set of TSP instances as previous section. The results are presented in the fourth rows of Table 3. It can be observed that the performance improves significantly. Comparing to using only n-gram models, it uses far less function evaluations, especially for larger instances, while retaining a high success rate. For completeness, we also include the results of using solely the distance-based heuristic for updating the population.

6 Discussion

The previous section showcased how we can use evolving mixture to incorporate other heuristics and prior knowledge about the problem into the search mechanism. The results also provide some support for our intuition that different methods or heuristics may be more suitable than others at different stages of the optimization process. For example, Fig. 2 shows the shifts of weights among three distributions in a run for solving `st70`. It illustrates how the evolving mixture adjusts the emphasis on different models and heuristics at different stages of the process. This adjustment is dynamically determined based on the promising solutions in the holdout set. We believe that this dynamic behavior may lead to a more efficient search strategy for finding the global optima.

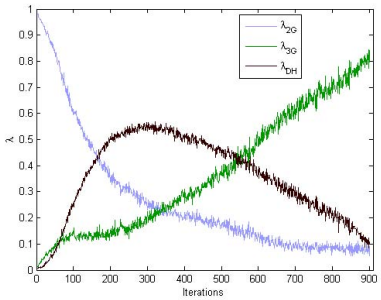

Fig. 2. Typical variation of weights associated with $P_{2\text{G}}$, $P_{3\text{G}}$ and P_{DH}. This illustration is from a run on `st70` with $N = 500$ and 30% of the population as holdout set.

Table 3. Experiment Results

(a) `ulysses16`

Method	Success Rate	# of Evaluations mean	std
2G	30/30	5461.3	902.4
3G	23/30	7207.0	10552.4
2G+3G	30/30	5040.0	1250.3
2G+3G+DH	30/30	4597.3	973.0
DH	30/30	17725.3	12070.3

(b) `gr24`

Method	Success Rate	# of Evaluations mean	std
2G	30/30	18436.0	2356.7
3G	25/30	16574.4	11117.6
2G+3G	30/30	16492.0	2325.1
2G+3G+DH	30/30	11432.0	1717.7
DH	30/30	13340.0	6265.2

(c) `bay29`

Method	Success Rate	# of Evaluations mean	std
2G	30/30	36332.0	3914.3
3G	22/30	40336.4	43365.6
2G+3G	30/30	31218.5	3409.1
2G+3G+DH	29/30	22785.0	3251.0
DH	19/30	107063.4	44051.6

(d) `att48`

Method	Success Rate	# of Evaluations mean	std
2G	27/30	298053.3	54469.6
3G	10/30	196176.0	127370.7
2G+3G	26/30	215021.5	18175.5
2G+3G+DH	30/30	111544.0	10272.6
DH	28/30	265637.1	90849.8

(e) `gr48`

Method	Success Rate	# of Evaluations mean	std
2G	30/30	277024.0	34535.4
3G	9/30	224640.0	118490.2
2G+3G	30/30	230048.0	23985.8
2G+3G+DH	30/30	154984.0	20243.5
DH	0/30	N/A	N/A

(f) `eil51`

Method	Success Rate	# of Evaluations mean	std
2G	19/30	493572.6	73781.0
3G	2/30	199665.0	39308.1
2G+3G	21/30	391182.1	90013.1
2G+3G+DH	29/30	217031.4	52819.1
DH	8/30	448513.1	157609.4

(g) `berlin52`

Method	Success Rate	# of Evaluations mean	std
2G	29/30	264276.6	74339.8
3G	9/30	320348.9	237082.3
2G+3G	29/30	217171.7	26798.3
2G+3G+DH	30/30	113906.0	9854.6
DH	30/30	180882.0	77478.7

(h) `st70`

Method	Success Rate	# of Evaluations mean	std
2G	30/30	1426938.3	201534.3
3G	12/30	554983.3	69040.0
2G+3G	29/30	930444.8	76827.8
2G+3G+DH	30/30	298841.7	16886.3
DH	0/30	N/A	N/A

(i) `kroA100`

Method	Success Rate	# of Evaluations mean	std
2G	30/30	3484900.0	208750.7
3G	7/30	2068357.1	233884.2
2G+3G	30/30	2773483.3	107248.1
2G+3G+DH	30/30	650783.3	46476.1
DH	0/30	N/A	N/A

(j) `lin105`

Method	Success Rate	# of Evaluations mean	std
2G	30/30	3772142.5	230364.5
3G	3/30	1984325.0	154667.8
2G+3G	28/30	2844131.3	119353.2
2G+3G+DH	30/30	572197.5	19629.8
DH	0/30	N/A	N/A

Table 4. Performance Comparison

(a) Solving `gr48`

Method	N	Success Rate	# of Evaluations mean	# of Evaluations std
OX	120	0/10	N/A	N/A
OX	960	2/10	287852	6706
eER	120	0/10	N/A	N/A
eER	960	5/10	166286	4932
PMX	120	0/10	N/A	N/A
PMX	960	0/10	N/A	N/A
EHBSA-WT	120	10/10	144032	29115
2G+3G	120	10/10	131724	16748
2G+3G+DH	120	10/10	83508	17105

(b) Solving `pr76`

Method	N	Success Rate	# of Evaluations mean	# of Evaluations std
OX	120	0/10	N/A	N/A
OX	960	0/10	N/A	N/A
eER	120	0/10	N/A	N/A
eER	960	3/10	394887	22321
PMX	120	0/10	N/A	N/A
PMX	960	0/10	N/A	N/A
EHBSA-WT	120	9/10	457147	65821
2G+3G	120	10/10	405660	54893
2G+3G+DH	120	10/10	195960	28123

Readers who are familiar with the Ant Colony Optimization (ACO) [16] might relate ACO to our approach in the aspect that they both use some distance-based heuristics to provide partial guidance for the search process. However, the crucial difference between the two is that ACO holds the heuristic term constant throughout a run with a user-specified parameter. On the other hand, our proposal dynamically adjusts the weights associated with the incorporated models and heuristics as the search proceeds. As mentioned previously, we think this dynamic adjustment may be more efficient than the static combination. Furthermore, it also simplifies some of the manual tuning associated with incorporating multiple models and heuristics.

7 Summary and Future Works

In this work, we have experimented with a set of EDAs that use probabilistic models estimated from n-gram statistics. To provide a smooth transition from lower-order model to higher-order ones, we proposed using a method that combines multiple models in the form of a linear combination. The weights associated with those models are estimated automatically from a reserved portion of the population. An additional advantage of this approach is that it also provides a convenient way to incorporate other heuristics and prior knowledge about the problem into the search mechanism.

As future work, we would like to compare our proposal to other approaches that also deal with sequencing problems. To provide some initial comparison, we adopt some experiment results from Tsutsui et al.'s work [8] which lists the performance of their method, EHBSA-WT[3], along with three other classical EA approaches, OX [17], eER [18] and PMX [19], on two TSP instances taken from TSPLIB. Note that EHBSA-WT gave the bset empirical performance on TSPs in the review by Ceberio et al. [11]. To compare our proposals with those approaches, we adopted their settings for running our algorithms. Table 4 shows the results of our methods along with the data taken from [8]. Note that in these experiments, we followed their setting which only performs ten runs per algorithm. For statistical significance, this might not be enough. So, the comparison should be taken merely as suggestive. However, the margin between our proposal and other methods seems to be quite prominent. Thus, we believe our proposal offers a promising direction for further investigation and development.

[3] A variant of EHBSA. More specifically, we adopt the results of configuration EHBSA-WT2, which is more similar to our proposal in the way of doing partial sampling.

References

1. Larrañaga, P., Lozano, J.A.: Estimation of Distribution Algorithms: A New Tool for Evolutionary Computation. Kluwer Academic Publishers (2001)
2. Pelikan, M., Sastry, K., Cantú-Paz, E.: Scalable optimization via probabilistic modeling: From algorithms to applications. Springer (2006)
3. Mendiburu, A., Miguel-Alonso, J., Lozano, J.A., Ostra, M., Ubide, C.: Parallel EDAs to create multivariate calibration models for quantitative chemical applications. Journal of Parallel and Distributed Computing 66(8), 1002–1013 (2006)
4. Chen, C.H., Chen, Y.P.: Real-coded ECGA for economic dispatch. In: Proceedings of the 9th GECCO, pp. 1920–1927. ACM (2007)
5. Ducheyne, E.I., De Baets, B., De Wulf, R.: Probabilistic models for linkage learning in forest management. In: Knowledge Incorporation in Evolutionary Computation, pp. 177–194. Springer (2005)
6. Robles, V., de Miguel, P., Larranaga, P.: Solving the traveling salesman problem with EDAs. In: Estimation of Distribution Algorithms, pp. 211–229. Springer (2002)
7. Tsutsui, S.: Probabilistic model-building genetic algorithms in permutation representation domain using edge histogram. In: Guervós, J.J.M., Adamidis, P.A., Beyer, H.-G., Fernández-Villacañas, J.-L., Schwefel, H.-P. (eds.) PPSN 2002. LNCS, vol. 2439, pp. 224–760. Springer, Heidelberg (2002)
8. Tsutsui, S., Pelikan, M., Goldberg, D.E.: Using edge histogram models to solve permutation problems with probabilistic model-building genetic algorithms. Technical Report 2003022 (2003)
9. Lozano, J.A., Mendiburu, A.: Solving job scheduling with estimation of distribution algorithms. In: Estimation of Distribution Algorithms: A New Tool for Evolutionary Computation, pp. 231–242. Kluwer Academic Publishers (2002)
10. Bean, J.C.: Genetic algorithms and random keys for sequencing and optimization. INFORMS Journal on Computing 6(2), 154–160 (1994)
11. Ceberio, J., Irurozki, E., Mendiburu, A., Lozano, J.A.: A review on estimation of distribution algorithms in permutation-based combinatorial optimization problems. Progress in Artificial Intelligence 1(1), 103–117 (2012)
12. Chuang, C.Y., Chen, Y.P.: On the effectiveness of distributions estimated by probabilistic model building. In: Proceedings of the 10th annual conference on Genetic and evolutionary computation, pp. 391–398. ACM (2008)
13. Beyer, H.G., Schwefel, H.P.: Evolution strategies–a comprehensive introduction. Natural Computing 1(1), 3–52 (2002)
14. Reinelt, G.: TSPLIB–a traveling salesman problem library. ORSA Journal on Computing 3(4), 376–384 (1991)
15. Jelinek, F., Mercer, R.L.: Interpolated estimation of markov source parameters from sparse data. In: Proceedings of the Workshop on Pattern Recognition in Practice (1980)
16. Dorigo, M., Gambardella, L.M.: Ant colony system: A cooperative learning approach to the traveling salesman problem. IEEE Transactions on Evolutionary Computation 1(1), 53–66 (1997)
17. Oliver, I.M., Smith, D.J., Holland, J.R.C.: A study of permutation crossover operators on the traveling salesman problem. In: Proceedings of the Second International Conference on Genetic Algorithms on Genetic Algorithms and Their Application, pp. 224–230. Erlbaum Associates Inc., Hillsdale (1987)
18. Starkweather, T., Mcdaniel, S., Whitley, D., Mathias, K., Whitley, C.: A comparison of genetic sequencing operators. In: Proceedings of the Fourth International Conference on Genetic Algorithms (1991)
19. Goldberg, D.E., Lingle, R.: Alleles, loci and the traveling salesman problem. In: Proceedings of the First International Conference on Genetic Algorithms and Their Applications, pp. 154–159 (1985)