Competitive Hopfield Network Combined With Estimation of Distribution for Maximum Diversity Problems

Jiahai Wang, *Member, IEEE*, Yalan Zhou, Jian Yin, and Yunong Zhang, *Member, IEEE*

***Abstract*—This paper presents a discrete competitive Hopfield neural network (HNN) (DCHNN) based on the estimation of distribution algorithm (EDA) for the maximum diversity problem. In order to overcome the local minimum problem of DCHNN, the idea of EDA is combined with DCHNN. Once the network is trapped in local minima, the perturbation based on EDA can generate a new starting point for DCHNN for further search. It is expected that the further search is guided to a promising area by the probability model. Thus, the proposed algorithm can escape from local minima and further search better results. The proposed algorithm is tested on 120 benchmark problems with the size ranging from 100 to 5000. Simulation results show that the proposed algorithm is better than the other improved DCHNN such as multistart DCHNN and DCHNN with random flips and is better than or competitive with metaheuristic algorithms such as tabu-search-based algorithms and greedy randomized adaptive search procedure algorithms.**

***Index Terms*—Combinatorial optimization problem, competitive Hopfield neural network (HNN), estimation of distribution, maximum diversity problem (MDP).**

I. Introduction

GIVEN A SET of n elements and a diversity measure or pairwise difference d_{ij} between elements i and j ($d_{ij} = d_{ji}$), with $d_{ij} > 0$ for $j \neq i$ and $d_{ij} = 0$ if otherwise, the maximum diversity problem (MDP) consists in selecting a subset of given cardinality m from n elements, such that the sum of the pairwise differences between the elements of the selected subset is maximized. Let $\nu_i = 1$ if element i belongs to the subset and $\nu_i = 0$ if otherwise. The MDP can be formulated as follows:

$$f(V) = \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} d_{ij}\nu_i\nu_j \tag{1}$$

subject to

$$\sum_{i=1}^{n} \nu_i = m. \tag{2}$$

The problem is also known as maximum dispersion [1], MAX-AVG dispersion [2], edge-weighted clique [3], remote clique [4], maximum edge-weighted subgraph [5], and dense k-subgraph [6].

There are several different applications of this model, for example, the allocation of available resources to preserve biological diversity [7], medical treatments, the scheduling of final exams, very large scale integration design, and data mining [8]. Duarte and Marti [9] proposed an interesting application of the MDP for evolutionary algorithms. The MDP would be solved to help the scatter search (SS) algorithm obtain a good solution set with a balance between quality and diversity. Furthermore, it is also a particularly complex and important issue in multiobjective evolutionary optimization to keep the diversity of the population. More details on the applications can be found in [4], [7]–[9].

The MDP is strong NP-hard [10]. The proposed exact algorithms [3], [11], [12] are able to solve only instances of size less than 50 variables in reasonable computation time. Several approximation algorithms with guaranteed performance ratios [2], [6], [13] were proposed for the MDP. However, numerical results for such algorithms are usually not provided. Thus, they are not compared to other algorithms to show how well they perform in practice.

Metaheuristics have shown to be very successful in solving many combinatorial optimization problems. Kincaid [14] firstly proposed a simulated annealing (SA) and a tabu search (TS) for the MDP. Simulation results on three sets of problem instances of size 25 show that the TS performs somewhat better than the SA. Following the general guidelines provided in [15], Macambira [5] proposed another TS algorithm, named multiple TS (MTS), for MDP instances with up to 100 vertices and found that the most difficult problem instances were obtained by taking $m = n/2$ from a computational standpoint. Alidaee *et al.* [16] proposed a different implementation of TS

Manuscript received June 5, 2008; revised August 27, 2008 and November 4, 2008. First published March 24, 2009; current version published July 17, 2009. This work was supported in part by the National Natural Science Foundation of China under Grants 60805026, 60573097, and 60773198, by the Guangdong Provincial Natural Science Foundation of China under Grant 07300630, by the Specialized Research Fund for the Doctoral Program of Higher Education under Grant 20070558052, by the Scientific Research Foundation for the Returned Overseas Chinese Scholars, State Education Ministry under Grant 2007-1108, and by the Program for New Century Excellent Talents in University under Grant NCET-07-0887.

J. Wang and J. Yin are with the Department of Computer Science, School of Information Science and Technology, Sun Yat-Sen University, Guangzhou 510275, China (e-mail: wjiahai@hotmail.com).

Y. Zhou is with the Information Science School, Guangdong University of Business Studies, Guangzhou 510320, China.

Y. Zhang is with the Department of Automation, School of Information Science and Technology, Sun Yat-Sen University, Guangzhou 510275, China.

Digital Object Identifier 10.1109/TSMCB.2008.2010220

using strategic oscillation for the MDP. However, simulation only for very small instances ($n < 50$) is given in [16].

A lot of greedy randomized adaptive search procedure (GRASP) methods were proposed for the MDP. In general, GRASP has two phases: solution construction phase and solution improvement or local search (LS) phase. The first GRASP was proposed by Ghosh [17] for MDP instances up to 40 elements. Andrade *et al.* [18] developed a new GRASP which can solve instances up to 250 elements and find better solutions on Ghosh's benchmark problems. Silva *et al.* [19] developed a series of GRASP algorithms by combining different constructions and LSs. These GRASPs were tested on a set of instances randomly generated up to 500 elements and obtained better solutions than previous GRASPs. In [20], Silva *et al.* further combined one of the GRASP algorithms with path-relinking (PR) technique. This hybrid, named KLD + PR, produced good solutions for instances of size up to 500. However, the KLD + PR requires a long computation time (e.g., about 10 h for $n = 500$; see [20]). Another GRASP with PR was also proposed by Andrade *et al.* [21].

GRASPs often mainly focus on the randomized generation of high quality starting solutions by very refined construction phases and sophisticated management of the solutions, while the subsequent solution improvement phase is usually performed by a rather simple LS. To explore an opposite method with respect to the GRASPs, i.e., to refine the LS phase while keeping a very simple initialization procedure, Aringhieri *et al.* [22] proposed a TS-based GRASP algorithm, named GRASP-TS in this paper. In the GRASP-TS, TS is initialized by a trivial constructive procedure, but it adds the tabu mechanism and suitable intensification and diversification devices to enhance the search in the improvement phase. Their simulation results show that the GRASP-TS achieves both better results and much shorter computational time with respect to those reported for the previous GRASPs [22]. They also extended their work by introducing further metaheuristics, such as variable neighborhood search (VNS) and SS [23], to enrich the improvement phase and yield a better exploration of the solution space. Simulation results show that the TS-based algorithm appears to be the best compromise between solution quality and performance.

Furthermore, Duarte and Martí [9] proposed a TS-based algorithm in which memory structures from the TS methodology are incorporated into both construction and improvement phase. Specifically, Duarte and Martí [9] proposed two new types of constructive algorithms (the first one is based on a GRASP construction, and the second one is based on memory structures) and combined them with three iterative improvement procedures: simple LS by Ghosh [17], improved LS, and short-term memory TS. They tested their algorithms on problem instances with up to 2000 vertices and compared the performance of 18 construction or improvement algorithms for the MDP. In particular, good performance was achieved by combining a constructive method based on the TS methodology with the short-term memory TS procedure. This hybrid algorithm, named Tabu_D2+LS_TS, outperforms previous methods in [9].

More recently, Palubeckis [24] proposed an iterated TS (ITS) for the MDP. The ITS also has two phases: solution perturbation and TS phases. Computational results for problem instances involving up to 5000 vertices show that the ITS provides a significantly better performance than previous algorithms. In particular, the ITS finds new best solutions for 69 test problems appearing in the literatures.

One possible and very promising approach to combinatorial optimization problems is to apply Hopfield neural networks (HNNs) [25]. In general, the HNN has two versions: continuous [26] and discrete [27] HNNs. For the continuous HNN, neurons have continuous values within (0, 1), and for the discrete HNN (DHNN), the neurons have only binary values—0 or 1. Takefuji, Funabiki, and their coworkers [28]–[32] found discrete neurons computationally more efficient than continuous neurons and therefore applied the DHNN to solve a variety of combinatorial optimization problems. The DHNN has an advantage over the continuous HNN in terms of the iteration of updating required to converge to a local minimum and the speed of executing each iteration.

Previous studies using neural network for the MDP are surprisingly scarce. Therefore, we propose a discrete competitive HNN (DCHNN) for the MDP in this paper. Adopting k-out-of-N rule, the DCHNN always provides a valid solution, and search space is greatly reduced without a burden on the parameter tuning. As the past research has expressed [28]–[32], however, the discrete form of the solution search can easily bring on the problem of the local minimum convergence. The neural network in the local minimum cannot move to other states, although the current state is not a good solution state. In order to deal with the local minima of the DHNN, some modified DHNN is proposed such as multistart DHNN [33], [34] and DHNN with random flips [35].

In the last decade, more and more researchers tried to overcome the drawbacks of usual recombination operators of evolutionary computation algorithms. Therefore, the estimation of distribution algorithms (EDAs) [36]–[45] have been developed. These algorithms, which have a theoretical foundation in probability theory, are also based on populations that evolve as the search progresses. The EDAs attempt to model the distribution of promising solutions and then produce the next generation by sampling the estimated distribution modeling. After every iteration, the distribution is reestimated.

In order to overcome the local minimum problem of DCHNN, the idea of EDA is combined with DCHNN, and a DCHNN-EDA is proposed for the MDP in this paper. Once the network is trapped in local minima, the perturbation based on EDA can generate a new starting point for the HNN for further search. It is expected that the further search is guided to a promising area by the probability model. Thus, the proposed DCHNN-EDA can escape from local minima and further search better results. The performance of the DCHNN-EDA is tested and evaluated by simulating a large number of benchmark instances. First, the proposed DCHNN-EDA is compared with other improved DHNNs. Simulation results show the superior performance of the DCHNN-EDA. We also mention the transiently chaotic neural network (TCNN) [46] and noisy chaotic neural network (NCNN) [47] proposed recently to deal with the local minima of the continuous HNN and discuss the similarities and differences between the chaotic neural models

and the DCHNN-EDA. Second, the proposed DCHNN-EDA is compared with metaheuristic algorithms. Simulation results show that the proposed DCHNN-EDA is better than or competitive with other metaheuristic algorithms such as ITS, MTS, GRASP-TS, Tabu_D2+LS_TS, and KLD+PR.

The contributions of this paper are as follows: 1) A novel competitive HNN method combined with EDA is proposed for the MDP and very good results are obtained, which is the contribution to the available literature on the MDP; 2) the probabilistic modeling/EDA idea is applied to HNN training and thus helps the network escape from local minima, which is the contribution to the HNN literature; and 3) a comprehensive experimental comparison of the proposed algorithm with other methods is also provided.

The remaining sections of this paper are organized as follows. In Section II, we propose a DCHNN for the MDP. In Section III, we propose a DCHNN combined with EDA for the MDP. In Section IV, benchmark data sets are used to evaluate the proposed DCHNN-EDA. The last section concludes this paper.

II. DCHNN FOR MDP

The MDP with an $n \times n$ symmetric matrix $\mathbf{D} = (d_{ij})$ can be mapped onto the HNN with n neurons. The ith neuron has input u_i and output ν_i. The output $\nu_i = 1$ if element i belongs to the subset, and $\nu_i = 0$ if otherwise. The energy function of the MDP can be written as

$$E = A\left(\sum_{i=1}^{n}\nu_i - m\right)^2 - \frac{B}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} d_{ij}\nu_i\nu_j \tag{3}$$

where A and B are the weighting factors.

The quality of the final solution is very sensitive to the values of these weighting factors (A and B). Therefore, a DCHNN is proposed to relieve this burden, in which the penalty term [the first term on the right-hand side of (3)] is handled in an explicit manner. In the DCHNN, a k-out-of-N competitive rule is imposed for the neuron updating. The k-out-of-N rule leads to a stable state with exactly k active neurons among N. Therefore, the neurons within the same row compete with one another to be fired. In other words, the input–output function for the neurons within the ith row is given by

$$\nu_i(t+1) = \begin{cases} 1, & \text{if } u_i(t) \geq u_{m\text{th}}(t) \\ 0, & \text{otherwise} \end{cases} \tag{4}$$

where $u_{m\text{th}}(t)$ is the mth largest value within $\{u_1(t), \ldots, u_n(t)\}$. The inputs u_i of all n neurons are sorted in a nonascending order. Then, from the beginning, m neurons among n neurons are selected as winner neurons. The outputs of those selected winner neurons are set to 1, and the outputs of other neurons are set to 0. If more than m neurons satisfy $u_i(t) \geq u_{m\text{th}}(t)$, neurons which have previously output 0 are selected with priority to be fired. This mechanism contributes to the blocking of continuous firings, encourages other neurons to be eventually fired, and ultimately assists the system to escape from a fixed point. Thus, search space is expanded by the state changes. The monotonicity property of the competitive neuron model (4) is equivalent to a MacCulloch–Pitts neuron model with a dynamic threshold equal to $u_{m\text{th}}(t)$.

By adopting this competitive model, the constraint of this problem described by (2) can always be satisfied. Therefore, the energy function consists of only one term that represents the sum of the pairwise differences between the elements of the selected subset since the constraint term is always satisfied

$$E = -\frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} d_{ij}\nu_i\nu_j. \tag{5}$$

According to the update rule of the DHNN [4], the inputs of the neurons for the MDP are computed by

$$\frac{du_i}{dt} = -\frac{\partial E}{\partial \nu_i} = \sum_{j=1}^{n} d_{ij}\nu_j. \tag{6}$$

However, in practice, they are approximated by the first-order Euler method in the form

$$\Delta u_i = \sum_{j=1}^{n} d_{ij}\nu_j. \tag{7}$$

Thus, the input u_i for each neuron of the DHNN is updated iteratively using the following equation based on the first-order Euler method [7]:

$$u_i(t+1) = u_i(t) + \Delta u_i. \tag{8}$$

One updating iteration means that all the n neurons have been updated once. The system with this kind of group updating way has a high convergence speed.

The DCHNN can thus be summarized as follows.

DCHNN for MDP
initialize **u** (randomly around zero);
calculate **v** using (4);
$t = 0$;
  repeat
    for $i = 1$ to n **do** /*HNN descent procedure*/
      compute $u_i(t+1)$ with motion equation (8);
    end for
    for $i = 1$ to n **do**
      update $\nu_i(t+1)$ using (4);
    end for
  $t = t + 1$;
until ($t > iterations$ or stability criteria are satisfied)

The discrete model has an advantage over the continuous model in terms of the number of updating required to converge to a local minimum and the speed of executing each iteration. However, the discrete form of the solution search can easily bring on the problem of the local minimum convergence. The neural network in the local minimum cannot move to other states. In the next section, we will propose a DCHNN combined with EDA for the MDP in order to overcome the local minimum problem of the DCHNN.

III. DCHNN Combined With EDA for MDP

This DCHNN can make significant advances in the earlier stages of optimization procedures (i.e., the energy function decreases rapidly and dramatically in the earlier stages), and improvements slow thereafter, until after several iterations, for example, ten iterations, a local minimum is encountered; thus, the energy function no longer decreases [34]. The local minimum problem is caused by the gradient descent dynamics of the update rule of the DHNN.

When the network is trapped in a local minimum, a perturbation operator is applied to the local minimum to generate a new starting point for the HNN. It is desirable that the generated starting point should be in a promising area in the search space. Therefore, in this section, we propose an EDA mutation operator as the perturbation operator in the HNN. The EDA mutation operator can generate a new starting point for the further HNN search. It is expected that the further search is guided to a promising area by the probability model. In the proposed algorithm, the solution mutation or perturbation is always applied to the current local minima.

In the following section, we first briefly review the EDA, and then, the DCHNN combined with the EDA is proposed.

A. EDA

EDA is a new area of evolutionary computation. In EDAs, there is neither crossover nor mutation operator. Instead, new population is generated by sampling the probability distribution which is estimated from the selected promising solutions or individuals of previous generation. Thus, these algorithms have a theoretical foundation in probability theory.

An algorithmic framework of most EDAs can be described as follows.

Framework of EDA
$Pop =$ InitializePopulation() /*Initialization*/
while Stopping criteria are not satisfied **do** /*Main Loop*/
$Pop_{sel} =$ Select(Pop) /*Selection*/
$Prob =$ Estimate(Pop_{sel}) /*Estimation*/
$Pop =$ Sample$(Prob)$ /*Sampling*/
endwhile

An EDA starts with a solution population Pop and a solution distribution model $Prob$. The main loop consists of three principal stages. The first stage is to select the best individuals (according to some fitness criteria) from the population. These individuals are used in the second stage in which the solution distribution model $Prob$ is updated or recreated. The third stage consists of sampling the updated solution distribution model to generate new solution offspring. There has been a growing interest for EDAs in the last year. A more comprehensive presentation of the EDA field can be found in [37] and [38].

B. DCHNN Combined With EDA

The EDA is a novel optimization tool and is primarily used in evolutionary algorithms which are population-based and multi-point search methods. However, a single EDA technique is hard for solving complicated optimization problems because the location information of solutions found so far (the actual positions of these solutions in the search space) is not directly used for the generation of offspring in the original EDA [41], [42]. Therefore, how to combine EDAs with other techniques represents an important research direction [43]. Recently, Zhang *et al.* [41]–[44] proposed several works on EDA hybrids for hard optimization problems. In particular, a new operator, called guided mutation, was proposed for generating new solutions. The guided mutation can be regarded as a combination of conventional mutation and EDA offspring generating scheme, and it was used both in population-based methods [41], [42] and single-point-based method [44]. Considering the local minima problem of the HNN and these successes recently reported in effectively combining EDA and other techniques [41]–[45], the EDA is introduced into the DCHNN, which is a single-point search method, to help the DCHNN escape from the local minima in this paper. The DCHNN combined with the EDA (DCHNN-EDA) can thus be summarized as follows.

DCHNN-EDA for MDP
initialize $\mathbf{u}$ (randomly around zero);
calculate $\mathbf{v}$ using (4);
initialize best-so-far solution $\mathbf{gb}$ and probability model $\boldsymbol{P}$;
for $k = 1$ to *descents* **do**
$t = 0$;
repeat
  for $i = 1$ to n **do** /*HNN descent procedure*/
    compute $u_i(t+1)$ with motion equation (8);
  end for
  for $i = 1$ to n **do**
    update $\nu_i(t+1)$ using (4);
  end for
$t = t + 1$;
until ($t > iterations$ or stability criteria are satisfied);
update and keep track for the best-so-far solution $\mathbf{gb}$;
update probability model using $\mathbf{v}$;
reset $\boldsymbol{P}$ to $1 - \boldsymbol{P}$ after a specified number of descents;
perturb the current solution $\mathbf{v}$ using EDA mutation to generate a new starting point $\mathbf{v}$ for the HNN;
reset $\mathbf{u}$ using (10);
end for

During the descents of HNN, the HNN will visit a number of locally optimal solutions. Statistical information of these optimal solutions can be extracted for building a probability model.

Several different probability models have been introduced into EDAs for modeling the distribution of promising solutions. The univariate marginal distribution (UMD) model is the simplest one and has been used in UMD algorithm [36], population-based incremental learning (PBIL) [39], and compact genetic algorithm [40]. Therefore, in this paper, the UMD model is adopted to estimate the distribution of good regions over the search space based on the locally optimal solutions. The proposed algorithm uses a probability vector $\boldsymbol{P} = (p_1, \ldots, p_i, \ldots, p_n)$ to characterize the distribution of promising

solutions in the search space, where p_i is the probability that the value of the ith position of a promising solution is 1.

Then, an EDA mutation mutates current solution based on the probability vector $\boldsymbol{P}$, which characterizes distribution of promising solutions. It can be expected that offspring falls in or close to a promising area in the search space. The EDA mutation perturbs the current solution or local minimum $\mathbf{v}$ and guides the HNN to search in binary 0–1 solution space in the following way.

Perturb the current solution using EDA mutation
for $i = 1$ to n **do**
  If rand() $< \beta$ /*Sample from probability vector*/ if rand() $< p_i$, set $\nu_i(t+1) = 1$, otherwise set $\nu_i(t+1) = 0$;
  Otherwise, $\nu_i(t+1) = \nu_i(t)$ /*Copy from current solution $\mathbf{v}^*$/
end for

where rand( ) produces a random number distributed uniformly on [0, 1]. In the EDA mutation, a bit is sampled from the probability vector $\boldsymbol{P}$ randomly or directly copied from the current solution $\mathbf{v}$, which is controlled or balanced by the parameter β. The larger the β, the more elements of the new starting point are sampled from the vector $\boldsymbol{P}$. Since some elements of offspring are sampled from the probability vector $\boldsymbol{P}$, it can be expected that they fall in or close to the promising area. The random sampling mechanism can also provide diversity for the search afterward.

The parameter β controls the strength of the perturbation or mutation. The mutation (kick move) implemented by the EDA mutation should be chosen adequately strong to allow one to leave the current local minimum and to enable the HNN to find new and possibly better solutions. If the kick move is too weak, the HNN may return after very few steps to the local minimum to which the kick move has been applied. On the other hand, the kick move should be adequately weak to keep the characteristics of the current local minimum, since part of the solution may already be close to optimal. A major problem for too strong kick move is that the resulting algorithm would be very similar to repeating the HNN search from the randomly generated solutions. Applying only rather small mutations has an additional advantage that the HNN requires only a few steps to reach the next local optimum, i.e., new local optima can be identified very fast—typically much faster than when starting from a randomly generated solution. Hence, for the same given computation time, more HNN searches can be performed than when starting from the randomly generated solutions.

In the proposed algorithm, the probability vector $\boldsymbol{P}$ is initialized as $\boldsymbol{P} = (0.5, \ldots, 0.5, \ldots, 0.5)$. The probability vector can be learned and updated at each descent of the HNN for modeling the distribution of promising solutions in the same way as in the PBIL algorithm [39]

$$p_i(t+1) = (1-\lambda)p_i(t) + \lambda\nu_i(t) \tag{9}$$

where $\lambda \in (0, 1]$ is the learning rate.

The main loop of the framework of EDA consists of three principal stages: selection, estimation, and sampling. In the DCHNN-EDA, current locally optimal solution $\mathbf{v}$ generated by the HNN descent procedure is selected; then, $\mathbf{v}$ is used in the estimation stage in which the probability model $\boldsymbol{P}$ is updated according to the rule defined by (9). The sampling process is described in the EDA mutation operator. However, there are several differences between the DCHNN-EDA and the most usual population-based EDA algorithms. First, in the sampling process of the DCHNN-EDA, the probability model is not sampled directly as in the pure or original EDA but rather used as part of the EDA mutation. Second, although the DCHNN-EDA uses a univariate probability model in the style of PBIL, only one solution is sampled from the probability model, and only one solution is used to update the model. Thus, the DCHNN-EDA is more akin to other EDA hybrids [41]–[45], particularly similar to the guided mutation used in iterated LS (ILS) for quadratic assignment problems (QAPs) [44]. The novelty of the DCHNN-EDA consists in applying the probabilistic modeling/EDA idea to train HNN and thus help the network escape from the local minima. In the future, we can also apply the DCHNN-EDA to the QAP and make a full comparison of the DCHNN-EDA with the ILS with guided mutation.

Guided by a probability model which characterizes the distribution of promising solutions in the search space, the EDA mutation mutates the current solution $\mathbf{v}$ to generate a new starting point. The EDA mutation operators provide a mechanism for combining global statistical information about the search space and the information of the current solution found during the previous search and thus generate new starting points for further search.

Before invoking the next HNN search, all the inputs of neurons u_i are renormalized or reset to 0 or 1 based on the current u_i values as follows [35]:

$$u_i = \begin{cases} 1, & \text{if } u_i > 0 \\ 0, & \text{otherwise.} \end{cases} \tag{10}$$

This prevents the inputs u_i of neurons from decreasing or increasing to too small or too large values according to (8) during successive evolution and therefore helps the states of neurons to be changed according to (4) in the next HNN search. Furthermore, after a specified number of descents, for example, five, without improvement of the best solution, it is possible to sample new solutions which are far from the current searching area by means of $1 - \boldsymbol{P}$.

The proposed algorithm repeatedly invokes two procedures—HNN search and EDA mutation—to construct a good starting solution for further HNN search. Parameter *descents* can be seen as an upper bound on the number of HNN invocations and then can be seen as a stopping criterion. One updating iteration is defined as one complete update of all n neurons according to (8) and (4). The total iteration number of the proposed algorithm is $descents \times iterations$.

For the MDP with a symmetric rational $n \times n$ matrix $\mathbf{D} = (d_{ij})$, the HNN uses n neurons. According to the motion function (6), the calculation in each neuron takes time $O(n)$.

TABLE I
COMPARISON OF THE RESULTS OBTAINED BY THE MULTISTART DCHNN, DCHNN WITH RANDOM FLIPPING PERTURBATION, AND DCHNN-EDA FOR SILVA INSTANCES

Instance	Best known value	DCHNN-EDA			multistart DCHNN			DCHNN with random flips		
		Best	Av.	Std.	Best	Av.	Std.	Best	Av.	Std.
Silva_100_10	333	**333**	333	0	328	317.00	8.07	**333**	322.17	6.69
Silva_100_20	1195	**1195**	1195	0	1178	1165.83	8.4	**1195**	1180.47	13.41
Silva_100_30	2457	**2457**	2457	0	2457	2384.37	23.02	**2457**	2419.10	32.82
Silva_100_40	4142	**4142**	4142	0	4137	4109.47	15.19	**4142**	4132.90	8.95
Silva_200_20	1247	**1247**	1244	3	1224	1204.00	11.6	1240	1223.60	11.27
Silva_200_40	4450	**4450**	4448.77	1.87	4410	4377.37	16.62	4446	4417.60	20.29
Silva_200_60	9437	**9437**	9435.67	4.99	9408	9342.2	31.85	**9437**	9403.50	34.72
Silva_200_80	16225	**16225**	16225	0	16184	16132.4	21.64	**16225**	16197.8	28.66
Silva_300_30	2694	**2694**	2685.77	4.94	2663	2623.93	17.41	2683	2647.87	19.89
Silva_300_60	9689	**9689**	9677	13.56	9591	9528.37	25.93	9667	9597.6	39.24
Silva_300_90	20743	**20743**	20733	13.91	20607	20553.93	28.65	20736	20664.27	44.43
Silva_300_120	35881	**35881**	35880.23	1.28	35839	35728.07	51.83	35873	35845.37	24.96
Silva_400_40	4658	**4658**	4655.37	3.3	4566	4538.30	14.97	4648	4578.97	33.62
Silva_400_80	16956	**16956**	16946.97	9.52	16794	16715.5	38.25	16910	16834.67	55.55
Silva_400_120	36317	**36317**	36313.9	6.71	36139	35994.47	57.78	36262	36170.4	65.17
Silva_400_160	62487	**62487**	62473.47	21.03	62233	62155.26	43.82	62487	62327.00	71.50
Silva_500_50	7141	**7141**	7127.8	6.3	7057	6967.97	30.2	7092	7033.57	37.19
Silva_500_100	26258	**26258**	26251.1	7.17	26042	25927.57	62.39	26230	26116.70	89.74
Silva_500_150	56572	**56572**	56572	0	56263	56151.27	48.22	56572	56444.73	91.62
Silva_500_200	97344	**97344**	97337.8	15.15	97090	96956.73	57.3	97323	97221.27	66.95

Therefore, sequential traversal of all n neurons takes time $O(n^2)$ in the proposed algorithm. Updating the probability model takes time $O(n)$. Therefore, the whole complexity of the proposed algorithm is $O(n^2)$.

IV. SIMULATION RESULTS AND DISCUSSIONS

In order to assess the performance of the proposed algorithm, simulations were implemented in C on a PC (Pentium4 2.80 GHz). In this section, we describe the benchmark data sets and the parameter setting, and present the results of the DCHNN-EDA on the benchmark data sets. We also compare the results with those HNN algorithms with different modifications to demonstrate the effect of the EDA perturbation and compare the results with other metaheuristics such as the ITS, MTS, GRASP-TS, Tabu_D2 + LS_TS, and KLD + PR.

A. Benchmark Data Sets

We tested the proposed algorithm on five data sets with a total of 120 instances. The size of the instances ranges from (small scale) 100 to (large scale) 5000. The name of instances and the best known solutions of 120 problems are shown in Tables I–XII. The details of the instance sets are described as follows [9], [24].

1) Silva instances: 20 $n \times n$ matrices with random integers generated from a [0, 9] uniform distribution with $n \in [100, 500]$ elements and $m \in [0.1n, 0.2n, 0.3n, 0.4n]$. These 20 instances are the largest instances first introduced by Silva *et al.* [19] and can be downloaded from http://www.ic.uff.br/~gsilva/instSilva.zip.
2) Random type 1 instances: matrices with real numbers generated from a (0, 10) uniform distribution. There are 20 instances (Type1_55) with $n = 500$ and $m = 50$, 20 instances (Type1_52) with $n = 500$ and $m = 200$, and 20 instances (Type1_22) with $n = 2000$ and $m = 200$.

   Note that instances Type1_55 are equal to instances Type1_52. They only differ in the amount of selected elements m. In other words, both instances are matrices of distances of 500 elements. In Type1_55, 50 elements are selected from 500 (i.e., $m = 50$), and in Type1_52, 200 elements are selected from 500 (i.e., $m = 200$).
3) Random type 2 instances: matrices with real numbers generated from a (0, 1000) uniform distribution. There are 20 instances with $n = 500$ and $m = 50$.

   Random type 1 and 2 instances are first introduced in [9] and can be downloaded from http://www.uv.es/~rmarti/paper/mdp.html.
4) Beasley instances: instances taken from the Operations Research (OR)-Library [48]. These test instances were originally introduced for the unconstrained binary quadratic optimization problem. Recently, these instances were adopted for testing algorithms for the MDP by Palubeckis [24]. In the case of MDP, the diagonal elements of these instances are ignored. All the instances have 10% density. In each case, the matrix contains both positive and negative numbers from [−100, 100]. There are ten instances with $n = 2500$ and $m = 1000$.

TABLE II
COMPARISON OF THE RESULTS OBTAINED BY THE MULTISTART DCHNN, DCHNN WITH RANDOM FLIPPING PERTURBATION, AND DCHNN-EDA FOR TYPE1_55 INSTANCES ($n = 500$; $m = 50$)

Instance	Best known value	DCHNN-EDA			multistart DCHNN			DCHNN with random flips		
		Best	Av.	Std.	Best	Av.	Std.	Best	Av.	Std.
Type1_55.1	7833.83	**7833.83**	7833.83	0	7749.4	7622.37	40.67	7821.89	7694.08	53.77
Type1_55.2	7771.66	**7771.66**	7750.68	19.82	7623.32	7574.62	21.07	7723.3	7645.59	40.15
Type1_55.3	7759.36	**7759.36**	7751.29	9.09	7652.39	7584.70	24.53	7757.13	7665.95	46.41
Type1_55.4	7770.24	**7770.24**	7759.94	11.85	7696.68	7585.61	32.22	7748.57	7664.47	41.34
Type1_55.5	7755.23	**7755.23**	7741.52	17.18	7620.6	7554.21	20.63	7740.06	7630.66	47.95
Type1_55.6	7773.71	**7773.71**	7771.42	1.79	7649.49	7592.31	23.14	7753.75	7656.89	48.53
Type1_55.7	7771.73	**7771.73**	7755.14	13.39	7651.25	7584.24	29.88	7742.77	7673.01	46.87
Type1_55.8	7750.88	7749.79	7738.6	10.71	7634.07	7582.29	22.24	7730.00	7655.37	33.35
Type1_55.9	7770.07	**7770.07**	7761.31	9.35	7683.59	7604.50	25.13	7744.44	7669.33	42.44
Type1_55.10	7780.35	**7780.35**	7773.01	10.62	7668.66	7589.94	26.44	7747.08	7680.70	33.44
Type1_55.11	7770.95	**7770.95**	7762.22	4.54	7650.28	7593.74	22.13	7736.05	7662.56	35.58
Type1_55.12	7757.65	**7757.65**	7748.63	1.91	7668.42	7587.65	31.55	7735.86	7647.30	51.02
Type1_55.13	7798.43	**7798.43**	7784.02	9.36	7652.12	7608.25	27.58	7757.41	7670.05	44.22
Type1_55.14	7795.63	**7795.63**	7780.58	16.86	7732.65	7602.49	34.80	7767.63	7657.66	47.92
Type1_55.15	7736.84	**7736.84**	7723.41	8.06	7616.58	7560.48	23.61	7710.75	7643.93	39.60
Type1_55.16	7792.77	**7792.77**	7785.76	10.65	7678.45	7602.74	24.16	7748.11	7679.52	37.55
Type1_55.17	7787.20	**7787.20**	7786.77	0.59	7699.08	7586.42	30.24	7740.02	7660.45	44.30
Type1_55.18	7756.26	**7756.26**	7745.96	11.52	7652.58	7576.34	25.48	7728.7	7642.62	40.51
Type1_55.19	7755.41	**7755.41**	7747.46	11.76	7667.13	7570.48	27.24	7744.45	7643.50	44.19
Type1_55.20	7733.86	**7733.86**	7723.58	7.78	7596.87	7561.2	21.53	7682.8	7629.16	28.72

TABLE III
COMPARISON OF THE RESULTS OBTAINED BY THE MULTISTART DCHNN, DCHNN WITH RANDOM FLIPPING PERTURBATION, AND DCHNN-EDA FOR TYPE1_52 INSTANCES ($n = 500$; $m = 200$)

Instance	Best known value	DCHNN-EDA			multistart DCHNN			DCHNN with random flips		
		Best	Av.	Std.	Best	Av.	Std.	Best	Av.	Std.
Type1_52.1	107394.58	**107394.58**	107387.59	7.6	107139.27	107061.59	43.78	107379.39	107311.24	48.92
Type1_52.2	107251.75	**107251.75**	107210.35	23.1	106953.72	106830.31	52.50	107179.42	107072.94	72.14
Type1_52.3	107260.39	**107260.39**	107258.72	0.89	106954.61	106864.96	48.68	107254.25	107151.41	69.77
Type1_52.4	107010.90	**107010.90**	107003.63	17.91	106663.00	106574.16	45.50	106984.99	106811.34	77.59
Type1_52.5	106944.55	**106944.55**	106926.69	12.92	106577.56	106473.15	53.03	106909.00	106744.07	89.26
Type1_52.6	107167.36	**107167.36**	107161.36	6.04	106845.80	106772.28	50.50	107155.42	107025.82	78.00
Type1_52.7	107079.44	**107079.44**	107078.64	0.15	106740.00	106623.88	49.39	107078.62	106888.78	107.19
Type1_52.8	107077.45	**107077.45**	107050.75	24.01	106764.67	106637.59	57.66	107060.17	106896.37	100.23
Type1_52.9	107482.71	**107482.71**	107480.12	2.57	107200.33	107068.25	61.47	107476.26	107406.83	85.44
Type1_52.10	107265.81	**107265.81**	107261.43	11.83	107002.41	106836.98	67.74	107231.46	107110.25	79.06
Type1_52.11	107193.08	**107193.08**	107191.18	4.85	106892.12	106801.73	42.57	107181.53	107091.31	61.18
Type1_52.12	106853.46	**106853.46**	106847.59	14.53	106555.21	106433.47	56.04	106835.60	106678.78	98.03
Type1_52.13	107647.28	**107647.28**	107645.94	2.07	107427.93	107303.55	48.99	107640.50	107568.42	54.97
Type1_52.14	107427.17	**107427.17**	107416.76	9.65	107118.75	106973.56	64.13	107421.48	107262.94	79.36
Type1_52.15	107054.79	**107054.79**	107040.47	13.09	106700.65	106588.28	49.99	107032.58	106877.27	98.50
Type1_52.16	107420.66	**107420.66**	107409.06	27.87	107106.63	106995.94	53.31	107366.37	107243.46	68.05
Type1_52.17	107111.01	**107111.01**	107101.94	13.93	106843.25	106727.10	55.27	107109.34	106983.15	59.81
Type1_52.18	107006.35	**107006.35**	106990.23	18.90	106716.21	106598.10	51.88	**107006.35**	106988.95	17.50
Type1_52.19	107052.95	**107052.95**	107031.41	6.97	106766.01	106663.31	56.03	107017.37	106922.99	68.53
Type1_52.20	106815.65	**106815.65**	106759.02	34.99	106425.71	106309.50	44.91	106723.21	106578.45	76.48

TABLE IV
COMPARISON OF THE RESULTS OBTAINED BY THE MULTISTART DCHNN, DCHNN WITH RANDOM FLIPPING PERTURBATION, AND DCHNN-EDA FOR TYPE1_22 INSTANCES ($n = 2000$; $m = 200$)

Instance	Best known value	DCHNN-EDA			multistart DCHNN			DCHNN with random flips		
		Best	Av.	Std.	Best	Av.	Std.	Best	Av.	Std.
Type1_22.1	114271	114209	114083.63	62.34	112319	112160.93	79.6	113590	113223.1	208.25
Type1_22.2	114327	114278	114043.43	80.47	112405	112128.27	107.16	113522	113135.4	237.25
Type1_22.3	114195	114126	113956.13	85.63	112469	112157.2	103.00	113657	113152.27	248.16
Type1_22.4	114093	114067	113853.2	79.06	112296	112053.17	96.38	113658	113059.63	235.11
Type1_22.5	114196	114070	113914.37	71.37	112228	112093.97	63.32	113501	113142.3	224.18
Type1_22.6	114265	114209	114084.83	67.92	112557	112213.4	114.43	113616	113266.93	193.36
Type1_22.7	114361	114317	114204.53	86.04	112509	112168.63	99.49	113625	113240.47	212.52
Type1_22.8	114327	114320	114063.33	85.79	112371	112152.07	87.16	113546	113156.03	191.74
Type1_22.9	114199	114165	113999.57	56.73	112465	112158.57	104.05	113591	113120.63	256.78
Type1_22.10	114229	114160	114033.7	83.58	112284	112108.43	91.12	113509	113166.17	233.90
Type1_22.11	114214	114166	113997.67	83.11	112291	112103.80	64.77	113811	113179.2	241.25
Type1_22.12	114214	114120	114000.57	74.02	112254	112095.00	91.22	113689	113199.07	225.86
Type1_22.13	114233	114144	113990.43	89.331	112359	112162.67	64.58	113671	113224.57	189.46
Type1_22.14	114216	114187	113986.58	118.75	112211	112033.5	83.52	113478	113078.13	187.02
Type1_22.15	114240	114194	114076.47	98.67	112428	112192.13	92.74	113810	113208.1	248.10
Type1_22.16	114335	114262	114126.77	82.26	112400	112210.60	99.88	113636	113255.4	256.55
Type1_22.17	114255	114177	114062.07	64.45	112404	112182.03	93.29	113594	113214.97	207.34
Type1_22.18	114408	114371	114243.23	54.29	112553	112307.83	85.91	113914	113253.17	281.56
Type1_22.19	114201	114158	114025.00	81.29	112153	111986.97	76.24	113419	113081.97	201.13
Type1_22.20	114349	114212	114090.47	79.24	112420	112229.90	83..80	113703	113219.96	203.41

TABLE V
COMPARISON OF THE RESULTS OBTAINED BY THE MULTISTART DCHNN, DCHNN WITH RANDOM FLIPPING PERTURBATION, AND THE DCHNN-EDA FOR TYPE2 INSTANCES ($n = 500$; $m = 50$)

Instance	Best known value	DCHNN-EDA			multistart DCHNN			DCHNN with random flips		
		Best	Av.	Std.	Best	Av.	Std.	Best	Av.	Std.
Type2.1	778030.57	**778030.57**	777639.36	754.16	767301.00	760454.27	2821.57	776378.48	767712.83	4599.98
Type2.2	779963.54	**779963.54**	779100.96	795.40	769917.00	760206.54	3441.80	775986.33	766248.4	5164.99
Type2.3	776768.17	**776768.17**	776030.42	630.99	767892.46	761058.45	2779.39	775561.82	766635.91	4682.16
Type2.4	775394.47	775383.61	775107.48	224.16	765467.86	758411.36	2552.29	774007.91	755220.59	3862.43
Type2.5	775610.96	**775610.96**	774702.39	1299.67	761933.62	756391.72	2799.52	774940.00	763958.40	3942.09
Type2.6	775153.58	**775153.58**	775090.50	339.71	765482.76	757585.99	2843.41	**775153.58**	764736.15	5539.65
Type2.7	777232.88	**777232.88**	776400.58	1031.17	765873.2	759353.86	2418.68	**777232.88**	767287.13	4291.83
Type2.8	779168.62	**779168.62**	778448.83	665.87	766465.92	760341.95	2784.07	777146.99	767830.33	4567.70
Type2.9	774802.05	**774802.05**	773834.11	815.22	763490.87	757962.21	2049.88	773777.65	765646.80	4245.99
Type2.10	774961.12	774696.92	774063.53	481.52	766038.99	756988.67	2828.35	773258.87	765410.44	4318.77
Type2.11	777468.78	**777468.78**	776569.82	825.24	764419.07	757513.69	3060.29	777468.78	765470.01	5037.42
Type2.12	775492.89	**775492.89**	774248.42	1111.04	758420.33	755199.65	2132.38	772011.45	764521.59	3747.66
Type2.13	780191.78	**780191.78**	780046.09	547.79	770016.35	760091.72	3223.43	776578.34	767050.26	3730.58
Type2.14	782232.68	**782232.68**	781900.27	765.83	769231.65	763301.05	2830.76	782232.68	771951.27	4625.79
Type2.15	780300.33	**780300.33**	780164.19	733.13	766239.62	759504.03	3070.40	773639.15	765113.12	3943.70
Type2.16	775436.19	**775436.19**	774847.89	664.19	761943.05	757045.51	2628.85	773485.72	765187.62	4223.38
Type2.17	776618.99	**776618.99**	774860.14	1259.10	764611.66	757071.27	3096.01	774239.18	764780.84	4359.80
Type2.18	775850.64	**775850.64**	775040.10	837.11	766008.7	758456.05	2832.05	772456.69	763810.70	5047.42
Type2.19	778802.82	**778802.82**	778284.97	566.11	768837.47	761596.41	3494.66	776837.18	768493.18	4400.24
Type2.20	778644.65	**778644.65**	776030.34	1881.79	763599.84	758839.51	2621.13	774826.22	766905.28	4836.76

TABLE VI
COMPARISON OF THE RESULTS OBTAINED BY THE DCHNN-EDA AND OTHER METAHEURISTIC ALGORITHMS FOR SILVA INSTANCES

Instance	Best known value	DCHNN-EDA		ITS		MTS		Tabu_D2+LS_TS	KLD+PR	GRASP-TS
		Best	Av.	Best	Av.	Best	Av.	Best	Best	Best
Silva_100_10	333	0	0	0	0	0	3.8	0	0	0
Silva_100_20	1195	0	0	0	0	0	8.6	0	0	0
Silva_100_30	2457	0	0	0	0	0	40.1	0	0	0
Silva_100_40	4142	0	0	0	0	0	2.5	0	0	0
Silva_200_20	1247	0	3	0	0	6	17.4	0	0	0
Silva_200_40	4450	0	1.23	0	0	0	16.6	0	0	0
Silva_200_60	9437	0	1.33	0	0	0	16.5	0	0	0
Silva_200_80	16225	0	0	0	0	0	14.7	0	0	0
Silva_300_30	2694	0	8.23	0	0	11	31.8	0	3	0
Silva_300_60	9689	0	12	0	3.2	0	41.2	0	0	0
Silva_300_90	20743	0	10	0	0	18	69.6	9	0	0
Silva_300_120	35881	0	0.77	0	0	3	20.7	3	2	0
Silva_400_40	4658	0	2.63	0	0	27	52.6	3	0	0
Silva_400_80	16956	0	9.03	0	0	24	78.7	8	11	0
Silva_400_120	36317	0	3.1	0	0	0	101.6	19	11	0
Silva_400_160	62487	0	13.53	0	7.0	10	88.1	31	0	0
Silva_500_50	7141	0	13.2	0	0	24	74.3	8	11	0
Silva_500_100	26258	0	6.9	0	0	15	96.8	4	4	0
Silva_500_150	56572	0	0	0	0	0	110.0	0	0	0
Silva_500_200	97344	0	6.2	0	0	18	108.7	0	0	0

TABLE VII
COMPARISON OF THE RESULTS OBTAINED BY THE DCHNN-EDA AND OTHER METAHEURISTIC ALGORITHMS FOR TYPE1_55 INSTANCES ($n = 500$; $m = 50$)

Instance	Best known value	DCHNN-EDA		ITS		MTS		Tabu_D2+LS_TS
		Best	Av.	Best	Av.	Best	Av.	Best
Type1_55.1	7833.83	0	0	0	0	46.82	118.86	0.01
Type1_55.2	7771.66	0	20.98	0	6.42	44.03	83.35	16.76
Type1_55.3	7759.36	0	8.07	0	0	2.30	66.54	9.72
Type1_55.4	7770.24	0	10.3	0	1.97	0	56.87	11.12
Type1_55.5	7755.23	0	13.71	0	2.85	49.25	102.08	6.72
Type1_55.6	7773.71	0	2.29	0	0	38.07	80.99	10.67
Type1_55.7	7771.73	0	16.59	0	0.84	22.11	67.35	19.03
Type1_55.8	7750.88	1.09	12.28	0	0	15.92	88.47	15.72
Type1_55.9	7770.07	0	8.76	0	2.45	9.38	59.70	16.14
Type1_55.10	7780.35	0	7.34	0	0	12.65	77.19	1.70
Type1_55.11	7770.95	0	8.73	0	2.10	49.07	91.43	1.34
Type1_55.12	7757.65	0	9.02	0	0	7.10	90.24	0
Type1_55.13	7798.43	0	14.41	0	0	54.90	108.80	14.63
Type1_55.14	7795.63	0	15.05	0	0	48.24	87.37	4.55
Type1_55.15	7736.84	0	13.43	0	3.20	29.59	82.38	18.13
Type1_55.16	7792.77	0	7.01	0	0	39.33	85.43	0
Type1_55.17	7787.20	0	0.43	0	0	0	78.44	1.22
Type1_55.18	7756.26	0	10.3	0	1.19	31.38	83.06	0
Type1_55.19	7755.41	0	7.95	0	0	58.53	101.03	0
Type1_55.20	7733.86	0	10.28	0	0	18.50	71.08	0

TABLE VIII
COMPARISON OF THE RESULTS OBTAINED BY THE DCHNN-EDA AND OTHER METAHEURISTIC ALGORITHMS FOR TYPE1_52 INSTANCES ($n = 500$; $m = 200$)

Instance	Best known value	DCHNN-EDA		ITS		MTS		Tabu_D2+LS_TS
		Best	Av.	Best	Av.	Best	Av.	Best
Type1_52.1	107394.58	0	6.99	0	0.68	26.82	59.53	-0.19
Type1_52.2	107251.75	0	41.4	0	0	82.13	152.68	95.76
Type1_52.3	107260.39	0	1.67	0	0	2.23	81.10	12.73
Type1_52.4	107010.90	0	7.27	0	0.63	17.77	119.15	24.60
Type1_52.5	106944.55	0	17.86	0	12.83	0	149.52	22.29
Type1_52.6	107167.36	0	6	0	1.15	55.45	156.64	2.83
Type1_52.7	107079.44	0	0.8	0	0	0.82	118.72	39.38
Type1_52.8	107077.45	0	26.7	0	14.66	1.50	113.60	63.21
Type1_52.9	107482.71	0	2.59	0	0	3.90	69.45	6.52
Type1_52.10	107265.81	0	4.38	0	0	29.23	121.58	70.06
Type1_52.11	107193.08	0	1.9	0	1.15	18.29	97.82	48.81
Type1_52.12	106853.46	0	5.87	0	5.25	20.78	112.29	40.73
Type1_52.13	107647.28	0	1.34	0	0	2.55	61.42	14.41
Type1_52.14	107427.17	0	10.41	0	3.34	8.36	124.92	23.30
Type1_52.15	107054.79	0	14.32	16.54	16.54	25.37	125.79	47.90
Type1_52.16	107420.66	0	11.6	0	0	5.31	151.35	50.64
Type1_52.17	107111.01	0	9.07	0	5.32	38.16	108.82	54.09
Type1_52.18	107006.35	0	16.12	0	2.43	24.25	166.63	52.71
Type1_52.19	107052.95	0	21.54	0	12.46	24.05	87.74	44.36
Type1_52.20	106815.65	0	56.63	0	6.54	106.67	224.12	80.51

TABLE IX
COMPARISON OF THE RESULTS OBTAINED BY THE DCHNN-EDA AND OTHER METAHEURISTIC ALGORITHMS FOR TYPE1_22 INSTANCES ($n = 2000$; $m = 200$)

Instance	Best known value	DCHNN-EDA		ITS		MTS		Tabu_D2+LS_TS
		Best	Av.	Best	Av.	Best	Av.	Best
Type1_22.1	114271	62	187.37	110	280.1	634	943.2	431
Type1_22.2	114327	49	283.57	75	293.2	652	1117.1	491
Type1_22.3	114195	69	238.87	69	242.1	709	957.1	664
Type1_22.4	114093	26	239.8	93	200.9	685	936.7	668
Type1_22.5	114196	126	281.63	171	362.7	652	822.9	780
Type1_22.6	114265	56	180.17	105	249.8	586	869.9	304
Type1_22.7	114361	44	156.47	33	208.3	721	988.9	660
Type1_22.8	114327	7	263.67	25	271.7	711	1039.4	599
Type1_22.9	114199	34	199.43	9	200.4	551	878.4	501
Type1_22.10	114229	69	195.3	166	266.9	618	829.7	658
Type1_22.11	114214	48	216.33	166	306.1	639	947.9	583
Type1_22.12	114214	94	213.43	124	283.4	548	919.7	566
Type1_22.13	114233	89	242.57	163	301.2	278	834.8	777
Type1_22.14	114216	29	229.42	147	355.0	589	1001.1	614
Type1_22.15	114240	46	163.53	11	138.2	403	809.8	390
Type1_22.16	114335	73	208.23	44	263.6	562	952.4	718
Type1_22.17	114255	78	192.93	187	260.7	707	916.9	498
Type1_22.18	114408	37	164.77	93	201.9	630	860.4	675
Type1_22.19	114201	43	176	118	281.0	777	974.8	643
Type1_22.20	114349	137	258.53	177	329.1	435	830.2	775

TABLE X
COMPARISON OF THE RESULTS OBTAINED BY THE DCHNN-EDA AND OTHER METAHEURISTIC ALGORITHMS FOR TYPE2 INSTANCES ($n = 500$; $m = 50$)

Instance	Best known value	DCHNN-EDA		ITS		MTS		Tabu_D2+LS_TS
		Best	Av.	Best	Av.	Best	Av.	Best
Type2.1	778030.57	0	391.21	0	0	5013.29	10039.02	140.44
Type2.2	779963.54	0	862.58	0	0	1018.11	9600.52	-0.27
Type2.3	776768.17	0	737.75	0	0	2958.72	8743.68	296.73
Type2.4	775394.47	10.86	286.99	0	0	524.02	8633.02	85.73
Type2.5	775610.96	0	908.57	0	329.84	4851.53	8466.06	-0.48
Type2.6	775153.58	0	63.08	0	0	983.37	8812.20	793.52
Type2.7	777232.88	0	832.3	0	0	1159.68	4868.34	844.13
Type2.8	779168.62	0	719.79	0	0	0	8161.97	945.12
Type2.9	774802.05	0	967.94	0	0	2879.18	6736.58	0.30
Type2.10	774961.12	264.2	897.59	0	105.68	1286.24	6419.93	1020.12
Type2.11	777468.78	0	898.96	0	263.99	3844.00	11566.61	518.47
Type2.12	775492.89	0	1244.47	0	0	5704.64	8981.63	2722.20
Type2.13	780191.78	0	145.69	0	197.62	2179.18	11166.08	-0.16
Type2.14	782232.68	0	332.41	0	398.76	2329.20	8345.42	576.74
Type2.15	780300.33	0	136.14	0	0	5108.39	11707.44	-0.30
Type2.16	775436.19	0	588.3	0	0	432.20	6718.14	0.13
Type2.17	776618.99	0	1758.85	0	557.87	2694.07	7400.73	1956.18
Type2.18	775850.64	0	810.54	0	57.56	2263.07	8840.64	349.58
Type2.19	778802.82	0	517.85	0	36.39	1612.60	8320.34	0.07
Type2.20	778644.65	0	2614.31	0	0	4149.72	10292.15	72.90

TABLE XI
COMPARISON OF THE RESULTS OBTAINED BY THE DCHNN-EDA AND OTHER METAHEURISTIC ALGORITHMS FOR BEASLEY INSTANCES ($m = 1000$)

Instance	Best known value	DCHNN-EDA			ITS		MTS	
		Best	Av.	Std.	Best	Av.	Best	Av.
b2500-1	1153068	1714	5735.33	1675.08	808	2550.2	8784	16487.8
b2500-2	1129310	1650	4223.07	1447.99	602	1251.6	11074	17601.0
b2500-3	1115538	2650	4590.93	1212.77	208	2074.4	13446	20356.4
b2500-4	1147840	1638	5457.87	1485.11	746	1688.6	12636	18488.8
b2500-5	1144756	1184	4448.13	1436.28	558	1211.4	13054	19598.6
b2500-6	1133572	832	5034.6	1830.56	250	1512.4	15942	20561.6
b2500-7	1149064	1700	3497.8	1195.34	306	1044.4	7586	15856.6
b2500-8	1142762	2498	4978.53	1497.51	324	1754.4	12850	17868.2
b2500-9	1138866	1720	4063.6	1523.48	810	2273.2	11506	21466.6
b2500-10	1153936	866	4218.6	1539.59	426	1457.8	11740	16726.8

In [24], Palubeckis introduced an MDP where constrain (2) was extended as follows:

$$m_1 \leq \sum_{i=1}^{n} \nu_i \leq m_2. \tag{11}$$

In the MDP defined by (1) and (11), the coefficients d_{ij} can be positive, negative, or zero. When all the coefficients d_{ij} are nonnegative, then (11) can be replaced by (2), where, obviously, $m_1 = m_2$. In [24], two experiments were conducted. In the first one, $m = m_1 = m_2 = 1000$ was set, whereas in the second, $m_1 = 1620$ and $m_2 = 1655$ were set. In this paper, only the MDP defined by (1) and (2) is considered, and therefore, we follow the first experiment in [24] when the Beasley instances are simulated. That is, the matrix $\mathbf{D}$ that contained both positive and negative elements is directly used in the DCHNN-EDA, as in [24], although we expect $d_{ij} > 0$, as stated in the MDP model defined by (1) and (11), which also facilitates the fair and direct comparison of

TABLE XII
COMPARISON OF THE RESULTS OBTAINED BY THE DCHNN-EDA AND OTHER METAHEURISTIC ALGORITHMS FOR RANDOM LARGER PROBLEMS ($n = 3000, 5000$; $m = 0.5n$)

Instance	Density	Best known value	DCHNN-EDA			ITS		MTS	
			Best	Av.	Std.	Best	Av.	Best	Av.
p3000-1	10	6501999	1331	2289.57	800.67	330	854.2	5675	8916.5
p3000-2	30	18272568	1373	3144.93	956.81	0	1124.3	11680	19699.4
p3000-3	50	29867138	1382	3622.93	1325.98	1271	2181.5	9472	16153.6
p3000-4	80	46914817	631	4121.03	1628.77	1159	2250.0	12521	18588.0
p3000-5	100	58095034	420	3004.6	1195.77	0	818.2	6096	14428.2
p5000-1	10	17508071	497	4015.43	1276.22	902	1920.7	11591	17426.4
p5000-2	30	50101514	2573	6532.07	1992.75	2205	279.2	21972	31588.9
p5000-3	50	82038723	3052	9558.07	3476.29	4331	6817.2	25795	37562.6
p5000-4	80	129411337	631	5751.2	2969.93	658	2705.7	26290	37354.4
p5000-5	100	160597469	2014	5782.87	2007.42	1370	3644.1	18037	28876.4

the results produced by the DCHNN-EDA with those produced by the ITS from [24].

5) Random larger instances: matrices with integer numbers generated from a [0, 100] uniform distribution. The densities of the matrix are 10%, 30%, 50%, 80%, and 100%. There are five instances with $n = 3000$ and $m = 0.5n$ and five instances with $n = 5000$ and $m = 0.5n$. These larger instances are first introduced in [24]. The sources of the generator and input files to replicate these problem instances can be found at http://www.soften.ktu.lt/~gintaras/max_div.html.

B. Parameter Setting

The DCHNN-EDA has two main parameters, which are parameter β and λ, to be tuned. The parameter β controls the strength of the mutation, and the learning rate λ balances the contributions between the old statistical information extracted from historical local minima and the information of the current local minimum to the new probability vector. The bigger the λ, the greater is the contribution of current local minimum.

To assess the effects of these parameters, the DCHNN-EDA is tested as a preexperimental tuning on a representative subset of problems with different sizes and densities from Tables I–XII. These selected problems include Silva_500_200, Type1_55.1, Type1_52.1, Type1_22.1, Type2.1, b2500-1, p3000-1, and p5000-5.

First, β and λ are defined to take values within the following discrete range $\{0.1, 0.2, \ldots, 0.9\}$, respectively. In the preliminary tests, we found that the parameter combinations $\beta \in \{0.1, 0.2, 0.3\}$ and $\lambda \in \{0.1\}$ can obtain good results, and there is no significant difference in the results using these parameter combinations. This suggests that relatively small mutation strength and contribution of the current local minimum are adequate for the MDP. The problem for too strong mutation strength and too big contribution of the current solution or local minimum is that the resulting algorithm is very similar to repeating the HNN search from randomly generated or initialized starting point like the multistart HNN [33].

Second, in order to further investigate the effect or sensitivity of parameter λ, we fix the parameter $\beta = 0.2$ and then vary parameter λ within the range $\{0.01, 0.02, \ldots, 0.09\}$. We find no significant difference in the results using different values of the parameter λ within the range, which suggests that the range of reasonable values of the parameter λ is rather large only if λ is set to a very small value. In fact, there is a more general reason for λ to be that low. Typically, in the PBIL, the probability vector is updated based on a histogram of solution variable values taken over a population of size larger than one, and thus, the potential increments to the probability vector are small (often smaller than one); therefore, λ is set on the order of 10^{-1}, for example, λ is set to 0.7 or 0.3 in [41] and [42]. However, in the DCHNN-EDA, since only one solution at a time is used to update to the probability vector according to (9), thus the potential increments to the probability vector are either 1 or 0, depending on the value of the current local minimum. Thus, the value of λ in the DCHNN-EDA should be smaller than that in the PBIL or, else, the new probability model will "forget" the old information extracted from the historical local minima quickly. Therefore, λ in the DCHNN-EDA can be recommended to set to the order of 10^{-2}.

Finally, β is randomly drawn from the interval [0.1, 0.3], and λ is randomly drawn from the interval [0.01, 0.1]; we also find no significant difference in the results.

Based on the preliminary computational experiments and analysis in theory on the parameters, we select a parameter combination $\beta = 0.2$ and $\lambda = 0.04$ for all test benchmark problems in the whole simulation. Certainly, the parameters are chosen experimentally, and the tuning of these parameters may be necessary when solving different optimization problems. In the future, we will research self-adaptive control strategy for these parameters.

The DCHNN for the MDP works hardest and most productively, measured by a rapidly decreasing energy, during the first ten iterations. Therefore, each descent in the DCHNN-EDA ends after ten iterations, and each test run invokes 500 times HNN search, i.e., $descents = 500$. The total iteration number of the DCHNN-EDA is thus $10^*500 = 5000$. This is a tradeoff between solution quality and computation time. If given more

computation time, the DCHNN-EDA can implement or invoke more times HNN search and therefore can be expected to produce better results.

C. Comparison With Other Modified DCHNN Algorithms

In this section, we first describe two modified DHNN algorithms proposed for combinatorial optimization problems [33], [35], and then, we compare the DCHNN-EDA with them to show how the EDA process can improve the performance of the DCHNN. Finally, we also mention the TCNN and NCNN proposed recently to deal with the local minima of the continuous HNN and point out similarities and differences between the chaotic neural models and the DCHNN-EDA.

He *et al.* [33] proposed a multistart HNN which repeats the random initial solution and HNN search mechanism a number of times and finally returns the best solution. He *et al.* applied the multistart DHNN to two-page crossing number and outerplanar drawing problems and obtained better results than other neural networks [33], [34]. We also propose a modified DCHNN which repeats the random initial solution and HNN search mechanism a number of times like in [33]. The structure of the multistart DCHNN algorithm is as follows.

Multistart DCHNN for MDP
initialize the best-so-far solution $\mathbf{gb}$;
for $k = 1$ to *descents* **do**
  initialize $\mathbf{u}$ (randomly around zero);
  calculate $\mathbf{v}$ using (4);
  $t = 0$;
  repeat
    for $i = 1$ to n **do** /*HNN descent procedure*/
      compute $u_i(t+1)$ with motion equation (8);
    end for
    for $i = 1$ to n **do**
      update $\nu_i(t+1)$ using (4);
    end for
  $t = t + 1$;
until ($t > iterations$ or stability criteria are satisfied);
update and keep track for the best-so-far solution $\mathbf{gb}$;
end for

In the multistart DCHNN, HNN descent procedure is restarted with different initial states of the neurons for several times, and finally, the best solution is output. In other words, the best local minimum among the local minima produced by several different descents of the HNN is selected as the best solution.

Smith *et al.* [35] proposed a modified HNN where random perturbation (random flip mutation) is used to help the network escape from local minima, and therefore, the HNN is called DHNN with random flips. The simulation results on timetabling problems showed that the DHNN with random flips is comparable to or better than the SA, TS, and greedy search methods [35]. We also introduce the random perturbation to the DCHNN. To avoid losing good solution during the evolution of the network, we add an additional step, update, and keep track for the best-so-far solution $\mathbf{gb}$. DCHNN with random flips can thus be summarized by the following.

DCHNN with random flips for MDP
initialize $\mathbf{u}$ (randomly around zero);
calculate $\mathbf{v}$ using (4);
initialize best-so-far solution $\mathbf{gb}$;
for $k = 1$ to *descents* **do**
  $t = 0$;
  repeat
    for $i = 1$ to n **do** /*HNN descent procedure*/
      compute $u_i(t+1)$ with motion equation (8);
    end for
    for $i = 1$ to n **do**
      update $\nu_i(t+1)$ using (4);
    end for
  $t = t + 1$;
  until ($t > iterations$ or stability criteria are satisfied);
  update and keep track for the best-so-far solution $\mathbf{gb}$;
  randomly choose a neuron, and flip according to threshold;
  reset $\mathbf{u}$ using (10);
end for

In the DCHNN with random flips, the stochasticity at the end of each iteration is created by randomly choosing a neuron and assigning its state value of 0 or 1. A threshold is used to control the stochasticity, so that if the threshold is 0.5, then there is an equal chance of a neuron state becoming 0 or 1. However, if the threshold is increased to, for example, 0.7, then there is only a 30% chance that the neuron will be assigned a value of 1. Thus, the modifications to the network dynamics described earlier contribute to an efficient wandering of the search space in short bursts of gradient descent, using random perturbations to escape local minima. In this paper, the threshold to control the stochasticity is set to 0.85, as in [35], and can obtain good performance of the method. Note that the stochasticity should be interpreted as a bias toward a neuron taking a value of 0 rather than a probability of a flip taking place.

The two modified HNNs mentioned earlier all invoke several times HNN search, which is similar to the DCHNN-EDA. Therefore, we compare the DCHNN-EDA with them in order to determine how much the *EDA process* contributes to the search.

Tables I–V show the best, average, and standard deviations produced by the multistart DCHNN, DCHNN with random flips, and the DCHNN-EDA, respectively, for the first three sets of test problem instances (including Silva instances and type 1 and 2 instances) in 30 independent runs. The best known values of these test problem instances are also shown in Tables I–V. From Tables I–V, we find that the DCHNN with random flips gets better, best, and average results than the multistart DCHNN. Furthermore, most of the average results of the DCHNN with random flips are better than the best results of the multistart DCHNN. It is apparent that the DCHNN-EDA performs better than the multistart DCHNN and DCHNN with random flips in all instances in terms of best and average values. Furthermore, most of the average results of the DCHNN-EDA are better than the best results of the multistart DCHNN and DCHNN with random flips.

TABLE XIII
ESTIMATED PERFORMANCES OF COMPUTERS

Computer	Performance estimated from Dongarra's paper (Mflop/s)	Estimated Dongarra's factor
AMD Athlon 1.4 GHz	704	0.5135
Pentium 4 3GHz	1571	1.1459
Pentium 4 Mobile 2.8GHz	-	1
Pentium M 1733 MHz notebook	-	1
Pentium 4 2.8GHz	1371	1

The multistart DCHNN cannot find any best known value. The DCHNN with random flips can find best known value in 9 out of 100 instances. The DCHNN-EDA can obtain a best known value in 77 out of 100 instances. Bold figures indicate the best known value obtained by all the algorithms.

In order to know whether the differences in performance showed in Tables I–V between the proposed DCHNN-EDA and DCHNN with random flips are statistically significant, unpaired t-tests ($sample\ size = 30$ and $degrees\ of\ freedom = 58$) were performed. From the two-tailed p value derived from the unpaired t-tests (for conciseness, p values are not shown in the tables because they can be easily obtained using TDIST function in Excel), we find that the difference of the average solution values between the two algorithms for all instances is extremely significant $(p < 0.0001)$ except for instance Type1_52.18 in Table III. For instance Type1_52.18, the difference of the average solution values between the two algorithms is not significant $(p = 0.7864)$.

In the multistart DCHNN, random initialization is a random sampling mechanism in the search space. That is, the starting solutions are generated completely randomly. If it is lucky enough to start from a good initial point, the multistart DCHNN can converge to a good solution. The DCHNN with random flips uses random perturbations to escape local minima. However, which neuron state is chosen to be flipped is decided in a completely random way, and therefore, it is somewhat "blind." In the DCHNN-EDA, the DCHNN is guided by global search information extracted from the EDA model and therefore can search better solution in the promising region, which is the main reason that the DCHNN-EDA can obtain better results than the multistart DCHNN and the DCHNN with random flips.

The multistart DCHNN, DCHNN with random flips, and the DCHNN-EDA use the same DHNN descent procedure, and they are different only in the construction of the starting point for each HNN descent. In the multistart DCHNN, the input and output of each neuron are randomly initialized at the beginning of each HNN descent. In the DCHNN-EDA, the probability model is updated, and the new starting point for further search is sampled from the probability model at the end of each descent. In the DCHNN with random flips, a bit is randomly chosen to flip at the end of each descent. Therefore, we can conclude that the three algorithms have about the same performance in running time. The computation time in terms of real CPU time of the DCHNN-EDA for all test instances is summarized in Tables XIII–XV.

In order to deal with the local minima of the continuous HNN, Chen and Aihara [46] proposed a TCNN by adding a decaying negative self-feedback to continuous HNN and thus introducing chaotic dynamics. Recently, Wang *et al.* [47] proposed a NCNN by adding decaying stochastic noise into the TCNN. In contrast to the TCNN and NCNN, the DCHNN-EDA does not introduce the stochastic or chaotic dynamics into the original HNN gradient descent dynamics and therefore does not attempt to prevent the system getting stuck at local minima. Rather, once the network converges to a local minimum, it aims to generate a new better starting point by EDA mutation for further HNN search. One advantage of this approach is that it has no effect on the energy minimization or original HNN gradient descent dynamics until the network converges to a stable point. In contrast to the TCNN and NCNN based on continuous HNN, the algorithm is based on DHNN and therefore is more efficient, fast, and simple. Furthermore, the DCHNN always provides a valid solution, and search space is greatly reduced without a burden on the parameter tuning. The DCHNN-EDA has also deterministic dynamics like in the TCNN and is not guaranteed to converge to a global optimum. However, from our simulation, the DCHNN-EDA can obtain good solutions within reasonable computation time.

D. Comparison With Metaheuristic Algorithms

In order to show the effectiveness of the DCHNN-EDA, we now present a comparative performance against other metaheuristic algorithms, including the ITS, MTS, Tabu_D2 + LS_TS, GRASP-TS, and KLD + PR. These algorithms are among the most accurate algorithms for the MDP. In particular, the ITS finds new best solutions for 69 test problems appearing in the literature and is the best algorithm for the MDP so far. Therefore, the ITS can be used as the benchmark algorithm to evaluate the performance of the DCHNN-EDA.

The main ingredients of the ITS include solution perturbation procedure, get start point, for construction of a starting solution, and a simple TS (STS) for iterative improvement of this solution. The ITS repeatedly invokes this two procedures. In the ITS for the MDP, the STS procedure contains only the main ingredient of TS, which is a short-term memory tabu list without an aspiration criterion. The STS consists of a best improvement LS and a short-term memory to escape from local minima and avoid cycles. The short-term memory is implemented as a tabu list that keeps track of the most recently visited solutions and forbids moves toward them, which is a good way of taking advantage of the history of the search. The complexity of the STS procedure in the ITS is $O(n^2)$. The perturbation procedure is to select variables and flip their values. These variables are randomly selected from a candidate list. The complexity of the perturbed operator in the ITS is

TABLE XIV
COMPARISON OF COMPUTATIONAL TIMES FOR SILVA INSTANCES

	KLD+PR		GRASP-TS	DCHNN-EDA	ITS MTS	Tabu_D2+LS_TS	
Running environments	C, PC 1.4 GHz AMD Athlon with 250 MB RAM.	NT	C, PC Intel Pentium 4 Mobile 2.8Ghz 512MB RAM	C, PC Pentium 4 2.8GHz 512MB	C, Pentium M 1733MHz notebook	Borland Builder 5.0, PC Pentium 4 3GHz with 1GB RAM	NT
Silva_100_10	11.0	5.65	-	0.95	20	10	11.5
Silva_100_20	28.4	14.58	-				
Silva_100_30	57.7	29.63	-				
Silva_100_40	89.7	46.06	-				
Silva_200_20	77.4	39.74	-	3.47			
Silva_200_40	321.3	164.99	5.35				
Silva_200_60	743.6	381.84	-				
Silva_200_80	1122.8	576.56	17.39				
Silva_300_30	413.7	212.43	13.09	7.58			
Silva_300_60	1842.7	946.23	39.40				
Silva_300_90	4183.1	2148.02	69.89				
Silva_300_120	7013.3	3601.33	93.18				
Silva_400_40	1300.6	667.85	45.20	12.90			
Silva_400_80	5957.1	3058.97	128.28				
Silva_400_120	13850.7	7112.33	208.22				
Silva_400_160	20456.7	10504.52	295.32				
Silva_500_50	3072.5	1577.73	94.33	18.24			
Silva_500_100	15857.2	8142.67	272.45				
Silva_500_150	34661.4	17798.63	446.43				
Silva_500_200	52497.4	26957.41	627.11				

TABLE XV
COMPARISON OF COMPUTATIONAL TIMES FOR TYPE1–2, bqp2500, AND RANDOM LARGER INSTANCE

	DCHNN-EDA	ITS, MTS	Tabu_D2+LS_TS	
Running environments	C, PC Pentium 4 2.8GHz 512MB	C, Pentium M 1733MHz notebook	Borland Builder 5.0, PC Pentium 4 3GHz with 1GB RAM	NT
Type1_55 (n=500, m=50)	18.21	20	10	11.5
Type1_52 (n=500, m=200)	18.26	20		
Type1_22 (n=2000, m=200)	358.35	20		
Type2 (n=500, m=50)	18.22	20		
bqp2500	452.62	1200	-	-
p3000	655.97	1200	-	-
p5000	1702.07	3600	-	-

$O(n^3)$. However, if the number of the selected variables satisfies a condition, then the complexity of perturbed operator decreases to $O(n^2)$. There are six parameters to be tuned in the ITS.

Tables VI–X display the name of the instances, the best known solution value, and the results of the DCHNN-EDA and the competitors. Similar to [24], we report the relative values of the solutions, i.e., "Best" means the deviation from the best known solution value of the best solution value found over 30 runs, and "Av." means the deviation from the best known solution value of the average solution value found over 30 runs. The results of the competitors are directly adopted from the original papers of the competitors.

From Table VI for Silva instances, we can find that the DCHNN-EDA obtains better solutions than the MTS, Tabu-D2 + LS-TS, and KLD + PR. The DCHNN-EDA, ITS, and GRASP-TS all can find the best known solutions for all Silva instances.

From Tables VII–X, we can find that the DCHNN-EDA outperforms the MTS and Tabu-D2 + LS_TS significantly and consistently. For Type1_55 (Table VII) and Type2 instances (Table X), the DCHNN-EDA is slightly worse than the ITS

in terms of both best and average solutions on the whole. For Type1_52 instances (Table VIII), the DCHNN-EDA is slightly better than the ITS in terms of best solution but slightly worse in terms of average solution on the whole. For Typ1_22 instances (Table IX), the DCHNN-EDA is better than the ITS in terms of both best and average solutions on the whole. Note that, in Tables VIII and X, some results for the Tabu-D2 + LS_TS are negative, which can be explained due to rounding errors in floating-point computations [24]. In the DCHNN-EDA, ITS, and MTS, all the real numbers are stored in variables of type "double."

E. Test for Larger Size Instances

In order to further show the performance of the DCHNN-EDA, the DCHNN-EDA is also tested on some larger size problems and compared with the ITS and MTS.

Tables XI and XII display the name of the instances, the best known solution value, and the results of the ITS, MTS, and DCHNN-EDA. For lager size instances, the absolute values of the solutions are very large. Therefore, we report the relative values of the solutions like in Tables V–X.

From Tables XI and XII, we can find that the DCHNN-EDA outperforms the MTS significantly and consistently for all the instances like in Tables VI–X. For Beasley and random larger instances, the DCHNN-EDA is worse than the ITS in terms of both best and average solutions. However, for random larger instances, the performance in terms of best solution seems to be very close to the ITS. Specially, the DCHNN-EDA can obtain better performance than the ITS in terms of best solution for p3000-4, p5000-1, p5000-3, and p5000-4 instances.

F. Comparison of Computation Time

A comparison of computation times for different heuristic algorithms is difficult because the different heuristic algorithms were tested on different computers. The computational efficiency of the heuristic algorithms is affected not only by the CPU but also by the implementation language, the complier, RAM, operating system, and even the programmer's skills. As done in many works [49], [50], the actual computation time from different computers is often converted or normalized to that on a benchmark computer for an approximate comparison. An indication of the machine's relative speed may be derived from the Mflop/s measure obtained by Dongarra [51]. Thus, the Mflop/s ratios called Dongarra's factors may be used to convert the CPU time of different machines. In this paper, we considered the same computer family for which performances are reported in Dongarra's paper. Table XIII presents the estimated performances and the Dongarra's factors that will be used to compare the computation time.

In Table XIII, we give the same factor for the Intel Pentium 4 Mobile 2.8 GHz with 512-MB RAM as our PC (Pentium4 2.80 GHz). We also test our algorithm on a Pentium M 1733 MHz notebook and obtain the similar performance as on our PC (Pentium4 2.80 GHz). Therefore, we give the same factor for the Pentium M 1733 MHz notebook as our PC (Pentium4 2.80 GHz).

Certainly, such factors lead to a crude computation time approximation. Lan and DePuy [49] pointed out that the average error associated with using the Dongarra conversion could exceed 50% in some cases. Furthermore, Johnson and McGeoch [52] pointed out that it is common that an algorithm may scale up with problem size differently, depending on machine types. Thus, Dongarra's factors, often used in OR, have to be considered as a rough normalization method and considered better than no normalization method at all. In this paper, we report both the actual computation time in different computers and the transformed or normalized time in second to our PC (Pentium4 2.80 GHz) in Tables XIV and V.

From Table XIV for Silva instances, we can find that the DCHNN-EDA is faster than other algorithms except for the Tabu_D2 + LS_TS. For Silva instances with 400 and 500 variables, the DCHNN-EDA is slightly slower than the Tabu_D2 + LS_TS. The KLD + PR is the most time-consuming algorithm and therefore cannot be used to solve larger problems. The GRASP + TS is also time consuming and thus difficult to be used to solve larger problems.

From Table XV, we can find that the DCHNN-EDA is faster than the ITS and MTS except for Type1_22 instances. The DCHNN-EDA is slower than the Tabu_D2 + LS_TS for Type1–2 instances.

In the ITS, the stopping criterion is based on the CPU clock. However, given a fixed execution time for the whole program, an open problem of the ITS is how to get optimal tradeoff between the number of TS restarts and the number of iterations of a TS run [24]. In the DCHNN-EDA, HNN search procedure can quickly reach a stable state within a constant number of iterations; therefore, the DCHNN-EDA can get good balance between the number of HNN searches and the solution quality more easily.

From all the results mentioned earlier, we can conclude that the DCHNN-EDA can obtain better or competitive results than metaheuristic algorithms within reasonable computation time.

G. Further Discussion in Multistart Framework

In this section, we discuss the DCHNN-EDA in the multistart algorithm framework and clarify the similarities and differences between the DCHNN-EDA and other metaheuristic algorithms.

The ITS, Tabu_D2 + LS_TS, GRASP-TS, KLD + PR, and the DCHNN-EDA can all be seen as multistart-based methods. In general, these methods have two phases: the solution construction phase in which the solution is generated and the solution improvement phase in which the solution is improved. Then, each global iteration produces a solution (usually local optima), and the best solution overall is the output of the algorithms [53]. In the solution construction phase, a simple strategy, for example, the random generation of a starting solution, can be used. Similarly, a simple solution improvement method can be applied, for example, simple LS. Furthermore, the solution improvement phase can be performed with a complex metaheuristic algorithm, for example, SA and TS.

The solution improvement phases of the ITS and GRASP-TS are all based on TS. In the Tabu_D2 + LS_TS, TS mechanism

is introduced into both solution construction and solution improvement phases to implement an explorative strategy. The KLD + PR combines the GRASP and PR strategy to improve the performance of the algorithm. As described in the previous sections, all these metaheuristic algorithms use more sophisticated procedures specifically tailored for the MDP. Misevièius *et al.* [54] pointed out that such "tailored" algorithms, like the ITS, Tabu_D2 + LS_TS, GRASP-TS, and KLD + PR, succeed in search if only they involve the specific problem knowledge. The algorithm designer must be very careful by implementing both the improvement and perturbation (construction) procedures. These procedures should be as much problem-oriented as possible. That is, these multistart-based algorithms are problem dependent, and the ideas and strategies implemented are difficult to apply directly to different problems.

In the DCHNN-EDA, the solution construction phase is based on EDA, and the solution improvement phase is based on HNN, which is a kind of simple and general LS. Thus, the DCHNN-EDA is based on a simple and efficient framework that can be used directly to design solving methods for other combinatorial optimization problems. In summary, the DCHNN-EDA is based on a much simpler and more general-purpose approach.

On the other hand, the motion equation of HNNs is based on gradient descent, but HNNs should not be viewed as naive gradient descent machines. They can be viewed as a machine which consists of a large number of simple interconnected processing units and therefore can implement complex computational task. Thus, the advantage of HNNs for combinatorial optimization problems is their inherently parallel structure and simple computational requirements, and therefore, HNNs are suitable for direct hardware implementation using analog or digital integrated circuit or optical computers [55], [56]. Considering the speed advantage, it is foreseeable that neural networks will soon become a rapid solution technique for large and complex combinatorial optimization problems [35].

Therefore, the DCHNN-EDA has the advantages of inherently parallel structure and simple computational requirements. In some cases, the DCHNN-EDA is not competitive with designer algorithms such as the ITS. Nevertheless, the aforementioned experimental results show that the DCHNN-EDA can obtain good results within reasonable computation time.

In order to further improve the performance of the DCHNN-EDA, we can borrow some ideas from the competitors or other metaheuristic algorithms by the following ways.

1) Adaptive perturbations. One possible method is to adopt the idea of the reactive search, which advocates the use of simple subsymbolic machine learning to automate the parameter tuning process and make it an integral part of the algorithm [57]. Another way of adapting the perturbation is to change deterministically its strength during the search, just as done in VNS [58].
2) Adaptive memory or mechanism to escape from local minima in the HNN descent. In the DCHNN-EDA, the solution improvement phase is based on a simple HNN search, and no history of the search or other mechanism is used to escape from local minima. Therefore, the tabu mechanism explicitly using the history of the search or other mechanism of escaping from local minima can also be incorporated into the HNN to further improve the performance of the DCHNN-EDA. Thus, the modified versions share similarities to the TS-based algorithms such as the ITS, Tabu_D2 + LS_TS, and GRASP-TS.
3) Population-based DCHNN-EDA extensions. Maintaining a population of solutions provides an alternative way to improve the exploration performance because the solutions in the population can explore different regions of the search space. In the simplest case, no interaction among the solutions or individuals of the population takes place. Certainly, variants that allow interaction among solutions or individuals can be introduced, which can further improve the performance of the algorithm. Thus, the population-based DCHNN-EDA extensions share similarities to the memetic algorithm [59].

V. Conclusion

In this paper, we have presented a novel DCHNN based on EDA as a competitive approach for the MDP compared with existing HNN algorithms and metaheuristic approaches. In order to overcome the local minimum problem of the DCHNN, the idea of EDA is incorporated into the DCHNN. In the DCHNN-EDA, the perturbation based on the EDA can generate a new starting point for the DCHNN for further search. Therefore, the DCHNN-EDA can escape from local minima and further search better results. Simulation results on the MDP show that the DCHNN-EDA is better than the other HNN algorithms such as multistart DCHNN and DCHNN with random flips and is better or competitive with metaheuristic algorithms such as the ITS, MTS, Tabu_D2 + LS_TS, GRASP-TS, and KLD + PR.

We also mention the TCNN and NCNN, and point out the similarities and differences between the chaotic neural models and the DCHNN-EDA. Furthermore, we discuss the DCHNN-EDA in the multistart algorithm framework and clarify the similarities and differences between the DCHNN-EDA and other metaheuristic algorithms. Based on these discussions, we highlight the characteristics of the DCHNN-EDA and propose some ways to further improve the performance of the DCHNN-EDA. At the same time, two future research lines can be followed. First, the DCHNN-EDA is a general optimization algorithm framework and therefore can be applied to other combinatorial optimization problems, for example, channel assignment [60], bipartite subgraph, maximum clique, maximum cut, and polygonal approximation [61] problems. Second, the perturbation based on EDA in the DCHNN-EDA can also be incorporated into other DHNN models, for example, hysteretic [62] and quantized [63], [64] HNNs, for solving different kinds of combinatorial optimization problems.

Acknowledgment

The authors would like to thank the anonymous associate editor and reviewers for the valuable suggestions and constructive comments.

REFERENCES

[1] M. J. Kuby, "Programming models for facility dispersion: The *p*-dispersion and maxisum dispersion problems," *Geogr. Anal.*, vol. 19, no. 4, pp. 315–329, Oct. 1987.

[2] S. S. Ravi, D. J. Rosenkrantz, and G. K. Tayi, "Heuristic and special case algorithms for dispersion problems," *Oper. Res.*, vol. 42, no. 2, pp. 299–310, Mar./Apr. 1994.

[3] E. M. Macambira and C. C. de Souza, "The edge-weighted clique problem: Valid inequalities, facets and polyhedral computations," *Eur. J. Oper. Res.*, vol. 123, no. 2, pp. 346–371, Jun. 2000.

[4] B. Chandra and M. M. Halldórsson, "Approximation algorithms for dispersion problems," *J. Algorithms*, vol. 38, no. 2, pp. 438–465, Feb. 2001.

[5] E. M. Macambira, "An application of tabu search heuristic for the maximum edge-weighted subgraph problem," *Ann. Oper. Res.*, vol. 117, no. 1–4, pp. 175–190, Nov. 2002.

[6] U. Feige, G. Kortsarz, and D. Peleg, "The dense *k*-subgraph problem," *Algorithmica*, vol. 29, no. 3, pp. 410–421, 2001.

[7] F. Glover, C. C. Kuo, and K. S. Dhir, "A discrete optimization model for preserving biological diversity," *Appl. Math. Model.*, vol. 19, no. 11, pp. 696–701, Nov. 1995.

[8] G. Kochenberger and F. Glover, *Diversity Data Mining*. Oxford, MS: Univ. Mississippi, 1999. Working Paper Series HCES-03-99.

[9] A. Duarte and R. Martí, "Tabu search and GRASP for the maximum diversity problem," *Eur. J. Oper. Res.*, vol. 178, no. 1, pp. 71–84, Apr. 2007.

[10] C. C. Kuo, F. Glover, and K. S. Dhir, "Analyzing and modeling the maximum diversity problem by zero–one programming," *Decis. Sci.*, vol. 24, no. 6, pp. 1171–1185, 1993.

[11] M. Hunting, U. Faigle, and W. Kern, "A Lagrangian relaxation approach to the edge-weighted clique problem," *Eur. J. Oper. Res.*, vol. 131, no. 1, pp. 119–131, May 2001.

[12] M. M. Sorensen, "New facets and a branch-and-cut algorithm for the weighted clique problem," *Eur. J. Oper. Res.*, vol. 154, no. 1, pp. 57–70, Apr. 2004.

[13] R. Hassin, S. Rubinstein, and A. Tamir, "Approximation algorithms for maximum dispersion," *Oper. Res. Lett.*, vol. 21, no. 3, pp. 133–137, Oct. 1997.

[14] R. K. Kincaid, "Good solutions to discrete noxious location problems via metaheuristics," *Ann. Oper. Res.*, vol. 40, no. 1–4, pp. 265–281, Dec. 1992.

[15] F. Glover, "Tabu search—Part I," *ORSA J. Comput.*, vol. 1, no. 3, pp. 190–206, 1989.

[16] B. Alidaee, F. Glover, G. Kochenberger, and H. Wang, "Solving the maximum edge weight clique problem via unconstrained quadratic programming," *Eur. J. Oper. Res.*, vol. 181, no. 2, pp. 592–597, Sep. 2007.

[17] J. B. Ghosh, "Computational aspects of the maximum diversity problem," *Oper. Res. Lett.*, vol. 19, no. 4, pp. 175–181, Oct. 1996.

[18] P. M. D. Andrade, A. Plastino, L. S. Ochi, and S. L. Martins, "GRASP for the maximum diversity problem," in *Proc. Fifth Metaheuristics Int. Conf.,* 2003, pp. 15-1–15-7. (CD-ROM).

[19] G. C. Silva, L. S. Ochi, and S. L. Martins, *Experimental Comparison of Greedy Randomized Adaptive Search Procedures for the Maximum Diversity Problem*, vol. 3059. Berlin, Germany: Springer-Verlag, 2004, pp. 498–512.

[20] G. C. Silva, M. R. Q. de Andrade, L. S. Ochi, S. L. Martins, and A. Plastino, "New heuristics for the maximum diversity problem," *J. Heuristics*, vol. 13, no. 4, pp. 315–336, Aug. 2007.

[21] P. M. D. Andrade, L. S. Ochi, and S. L. Martins, "GRASP with path-relinking for the maximum diversity problem," in *Proc. 4th Int. Workshop Efficient* Exp. *Algorithms WEA*, S. Nikoletseas, Ed. Berlin, Germany: Springer-Verlag, 2005, vol. 3539, pp. 558–569.

[22] R. Aringhieri, R. Cordone, and Y. Melzani, "Tabu search versus GRASP for the maximum diversity problem," *4OR: Q. J. Oper. Res.*, vol. 6, no. 1, pp. 45–60, Mar. 2008.

[23] R. Aringhieri and R. Cordonein "Better and faster solutions for the maximum diversity problem," DTI, Univ. Milano, Milan, Italy, Note del Polo 93, Jun. 2006. [Online]. Available: http://air.unimi.it/bitstream/2434/25779/1/notaDelPolo93-AringhieriCordone.pdf

[24] G. Palubeckis, "Iterated tabu search for the maximum diversity problem," *Appl. Math. Comput.*, vol. 189, no. 1, pp. 371–383, Jun. 2007.

[25] J. J. Hopfield and D. W. Tank, "'Neural' computation of decisions in optimization problems," *Biol. Cybern.*, vol. 52, no. 3, pp. 141–152, Jul. 1985.

[26] J. J. Hopfield, "Neurons with graded response have collective computational properties like those of two-state neurons," *Proc. Nat. Acad. Sci. U.S.A.*, vol. 81, no. 10, pp. 3088–3092, May 1984.

[27] J. J. Hopfield, "Neural networks and physical systems with emergent collective computational abilities," *Proc. Nat. Acad. Sci. U.S.A.*, vol. 79, no. 8, pp. 2554–2558, Apr. 1982.

[28] N. Funabiki, Y. Takefuji, and K. C. Lee, "Comparisons of seven neural network models on traffic control problems in multistage interconnection networks," *IEEE Trans. Comput.*, vol. 42, no. 4, pp. 497–501, Apr. 1993.

[29] N. Funabiki and Y. Takefuji, "A neural network parallel algorithm for channel assignment problems in cellular radio networks," *IEEE Trans. Veh. Technol.*, vol. 41, no. 4, pp. 430–437, Nov. 1992.

[30] N. Funabiki and J. Kitamichi, "A gradual neural network algorithm for broadcast scheduling problems in packet radio networks," *IEICE Trans. Fundam.*, vol. E82-A, no. 5, pp. 815–824, May 1999.

[31] N. Funabiki and S. Nishikawa, "A binary Hopfield neural-network approach for satellite broadcast scheduling problems," *IEEE Trans. Neural Netw.*, vol. 8, no. 2, pp. 441–445, Mar. 1997.

[32] N. Funabiki and S. Nishikawa, "A gradual neural-network approach for frequency assignment in satellite communication systems," *IEEE Trans. Neural Netw.*, vol. 8, no. 6, pp. 1359–1370, Nov. 1997.

[33] H. He, O. Sykora, and E. Makinen, "An improved neural network model for the two-page crossing number problem," *IEEE Trans. Neural Netw.*, vol. 17, no. 6, pp. 1642–1646, Nov. 2006.

[34] H. He and O. Sykora, "A Hopfield neural network model for the outerplanar drawing problem," in *Proc. IMECS*, Hong Kong, Jun. 20–22, 2006, pp. 42–47.

[35] K. A. Smith, D. Abramson, and D. Duke, "Hopfield neural networks for timetabling: Formulations, methods, and comparative results," *Comput. Ind. Eng.*, vol. 44, no. 2, pp. 283–305, Feb. 2003.

[36] H. Muhlenbein and G. Paaß, "From recombination of genes to the estimation of distributions," in Proc. *4th* Conf. *PPSN*, H.-M. Voigt, W. Ebeling, I. Rechenberg, and H.-P. Schwefel, Eds. Berlin, Germany: Springer-Verlag, 1996, vol. 1411, pp. 178–187.

[37] P. Larranaga and J. Lozano, "Estimation of distribution algorithms: A new tool for evolutionary computation," in *Genetic Algorithms and Evolutionary Computation*, vol. 2. New York: Springer-Verlag, 2001.

[38] M. Pelikan, D. E. Goldberg, and F. Lobo, "A survey of optimization by building and using probabilistic models," *Comput. Optim. Appl.*, vol. 21, no. 1, pp. 5–20, Jan. 2002.

[39] S. Baluja, "Population-based incremental learning: A method for integrating genetic search based function optimization and competitive learning," School Comput. Sci., Carnegie Mellon Univ., Pittsburgh, PA, Tech. Rep. CMU-CS-94-163, 1994.

[40] G. R. Harik, F. G. Lobo, and D. E. Goldberg, "The compact genetic algorithm," *IEEE Trans. Evol. Comput.*, vol. 3, no. 4, pp. 287–297, Nov. 1999.

[41] Q. Zhang, J. Sun, and E. Tsang, "An evolutionary algorithm with guided mutation for the maximum clique problem," *IEEE Trans. Evol. Comput.*, vol. 9, no. 2, pp. 192–200, Apr. 2005.

[42] Q. Zhang, J. Sun, G. Xiao, and E. Tsang, "Evolutionary algorithms refining a heuristic: A hybrid method for shared-path protections in WDM networks under SRLG constraints," *IEEE Trans. Syst., Man, Cybern. B, Cybern.*, vol. 37, no. 1, pp. 51–61, Feb. 2007.

[43] Q. Zhang, J. Sun, and E. Tsang, "Combinations of estimation of distribution algorithms and other techniques," *Int. J. Autom. Comput.*, vol. 4, no. 3, pp. 273–280, Jul. 2007.

[44] Q. Zhang and J. Sun, "Iterated local search with guided mutation," in *Proc. IEEE* Congr. Evol. Comput., Jul. 2006, pp. 924–929.

[45] P. Guturu and R. Dantu, "An impatient evolutionary algorithm with probabilistic tabu search for unified solution of some NP-hard problems in graph and set theory via clique finding," *IEEE Trans. Syst., Man, Cybern. B, Cybern.*, vol. 38, no. 3, pp. 645–666, Jun. 2008.

[46] L. Chen and K. Aihara, "Chaotic simulated annealing by a neural network model with transient chaos," *Neural Netw.*, vol. 8, no. 6, pp. 915–930, 1995.

[47] L. Wang, W. Liu, and H. Shi, "Noisy chaotic neural networks with variable thresholds for the frequency assignment problem in satellite communications," *IEEE Trans. Syst., Man, Cybern. C, Appl. Rev.*, vol. 38, no. 2, pp. 209–217, Mar. 2008.

[48] J. E. Beasley, *OR-Library: Unconstrained Binary Quadratic Programming (Beasley, 1990 and Badics, 1996)*. (5/11/2008). [Online]. Available: http://mscmga.ms.ic.ac.uk/jeb/orlib/bqpinfo.html

[49] G. Lan and G. W. DePuy, "On the effectiveness of incorporating randomness and memory into a multi-start metaheuristic with application to the set covering problem," *Comput. Ind. Eng.*, vol. 51, no. 3, pp. 362–374, Nov. 2006.

[50] J.-C. Créput and A. Koukam, "A memetic neural network for the Euclidean traveling salesman problem," *Neurocomput.*, 2008. doi: 10.1016/j.neucom.2008.01.023.

[51] J. J. Dongarra, "Performance of various computers using standard linear equations software," Dept. Comput. Sci., Univ. Tennessee, Knoxville, TN, Tech. Rep. CS-89-85, 2007. [Online]. Available: http://www.netlib.org/benchmark/performance.ps
[52] D. S. Johnson and L. A. McGeoch, "Experimental analysis of heuristics for the STSP," in *Traveling Salesman Problem and Its Variations*, G. Gutin and A. Punnen, Eds. Dordrecht, The Netherlands: Kluwer, 2002, pp. 369–443.
[53] R. Martí, "Multi-start methods," in *Handbook of Metaheuristics*, F. W. Glover and G. A. Kochenberger, Eds. Berlin, Germany: Springer-Verlag, 2003, pp. 355–368.
[54] A. Misevièius, J. Smolinskas, and A. Tomkevièius, "Using iterated tabu search for the traveling salesman problem: New results," *Inf. Technol. Control*, vol. 34, no. 4, pp. 327–337, 2005.
[55] D. L. Huang and J. Wang, "Digital hardware realization of a recurrent neural network for solving the assignment problem," *Neurocomputing*, vol. 51, pp. 447–461, Apr. 2003.
[56] E. Dominguez and J. Munoz, "A neural model for the *p*-median problem," *Comput. Oper. Res.*, vol. 35, pp. 404–416, 2008.
[57] R. Battiti and M. Brunato, "Reactive search: Machine learning for memory-based heuristics," in *Handbook of Approximation Algorithms and Metaheuristics*, T. Gonzalez, Ed. London, U.K.: Chapman & Hall, May 2007.
[58] P. Hansen and N. Mladenovic, "An introduction to variable neighborhood search," in *Meta-Heuristics: Advances and Trends in Local Search Paradigms for Optimization*, S. Voss, S. Martello, I. H. Osman, and C. Roucairol, Eds. Boston, MA: Kluwer, 1999, pp. 433–458.
[59] N. Krasnogor and J. E. Smith, "A tutorial for competent memetic algorithms: Model, taxonomy, and design issues," *IEEE Trans. Evol. Comput.*, vol. 9, no. 5, pp. 474–488, Oct. 2005.
[60] K. Ikenaga, Y. Takenaka, and N. Funabiki, "An expended maximum neural network algorithm for a channel assignment problem in cellular radio networks," *Electron. Commun. Jpn.*, vol. 83, no. 11, pp. 11–19, 2000.
[61] G. Galán-Marín and J. Muñoz-Pérez, "Design and analysis of maximum Hopfield networks," *IEEE Trans. Neural Netw.*, vol. 12, no. 2, pp. 329–339, Mar. 2001.
[62] G. Xia, Z. Tang, Y. Li, and J. Wang, "A binary Hopfield neural network with hysteresis for large crossbar packet-switches," *Neurocomputing*, vol. 67, pp. 417–425, Aug. 2005.
[63] S. Matsuda, "Quantized Hopfield networks for integer programming," *Syst. Comput. Jpn.*, vol. 30, no. 6, pp. 1354–1364, 1999.
[64] M. Nourelfath and N. Nahas, "Quantized Hopfield networks for reliability optimization," *Reliab. Eng. Syst. Saf.*, vol. 81, no. 2, pp. 191–196, Aug. 2003.

Jiahai Wang (M'07) received the B.S. degree from the Gannan Teachers College, Ganzhou, China, in 1999, the M.S. degree from Shandong University, Jinan, China, in 2001, and the Ph.D. degree from Toyama University, Toyama, Japan, in 2005.

In 2005, he joined Sun Yat-Sen University, Guangzhou, China, where he is currently an Instructor with the Department of Computer Science, School of Information Science and Technology. His main research interests include optimization theory, algorithms, and applications based on computational intelligence.

Yalan Zhou received the B.S. degree in computer science from the Guangdong University of Technology, Guangzhou, China, in 2003, the M.S. degree in software engineering from the South China University of Technology, Guangzhou, in 2005, and the Ph.D. degree from Sun Yat-Sen University, Guangzhou, in 2008.

In 2008, she joined the Guangdong University of Business Studies, Guangzhou, where she is currently an Instructor in the Information Science School. Her research interests include artificial intelligence and data mining.

Jian Yin received the B.S., M.S., and Ph.D. degrees in computer science from Wuhan University, Wuhan, China, in 1989, 1991, and 1994, respectively.

In 1994, he joined Sun Yat-Sen University, Guangzhou, China, where he is currently a Professor in the Department of Computer Science, School of Information Science and Technology. He has published more than 100 refereed journal and conference papers. His current research interests are in the areas of data mining, artificial intelligence, and machine learning.

Dr. Yin is a Senior Member of the China Computer Federation.

Yunong Zhang (M'02) received the B.S. degree from the Huazhong University of Science and Technology, Wuhan, China, in 1996, the M.S. degree from the South China University of Technology, Guangzhou, China, in 1999, and the Ph.D. degree from The Chinese University of Hong Kong, Shatin, Hong Kong, in 2003.

He is currently a Professor with the Department of Automation, School of Information Science and Technology, Sun Yat-Sen University (SYSU), Guangzhou. Before joining SYSU in 2006, he has been with the National University of Ireland, Dublin, Ireland; the University of Strathclyde, Glasgow, U.K.; and the National University of Singapore, Singapore, Singapore, since 2003. His main research interests include neural networks, robotics, and Gaussian process and computation.