Expert Systems with Applications 37 (2010) 5758–5774

Contents lists available at ScienceDirect

Expert Systems with Applications

journal homepage: www.elsevier.com/locate/eswa

Discrete Hopfield network combined with estimation of distribution for unconstrained binary quadratic programming problem

Jiahai Wang

Department of Computer Science, Sun Yat-sen University, No. 132, Waihuan East Road, Guangzhou Higher Education Mega Center, Guangzhou 510006, PR China

ARTICLE INFO

Keywords:
Discrete Hopfield neural network
Estimation of distribution
Unconstrained binary quadratic programming problem
Combinatorial optimization problem

ABSTRACT

Unconstrained binary quadratic programming problem (UBQP) consists in maximizing a quadratic 0-1 function. It is a well known NP-hard problem and is a unified model for a variety of combinatorial optimization problems. This paper presents a discrete Hopfield neural network (DHNN) combined with estimation of distribution algorithm (EDA) for the UBQP. The idea of EDA is combined with the DHNN in order to overcome the local minima problem of the network. Once the network is trapped in local minima, the perturbation based on EDA can generate a new starting point for the DHNN for further search, which is in a promising area characterized by a probability model. Thus, the proposed algorithm, named DHNN–EDA, can escape from local minima and further search better results. The DHNN–EDA is tested on a large number of benchmark problems with size up to 7000 variables. Simulation results on the UBQP show that the DHNN–EDA is better than the other improved DHNN algorithms such as multi-start DHNN and DHNN with random flips, and is better than or competitive with metaheuristic algorithms such as simulated annealing, tabu search, scatter search and memetic algorithm.

1. Introduction

In the unconstrained binary quadratic programming problem (UBQP), a symmetric rational $n \times n$ matrix $Q = (q_{ij})$ is given, and a binary vector $X = \{x_1, \ldots, x_i, \ldots, x_n\}$, $x_i \in \{0,1\}$ is searched, such that the objective function

$$f(X) = \sum_{i=1}^{n} \sum_{j=1}^{n} q_{ij} x_i x_j, \tag{1}$$

is maximized. Since for each binary variable $x_i = (x_i)^2$ (as $x_i \in \{0,1\}$), the objective function Eq. (1) can be written as:

$$f(X) = \sum_{i=1}^{n} \sum_{j=1 j \neq i}^{n} q_{ij} x_i x_j + \sum_{i=1}^{n} q_{ii} x_i, \tag{2}$$

where q_{ij} $(j \neq i)$ is the coefficient of the quadratic term and q_{ii} is the coefficient of the linear term. This problem is also known as the (unconstrained) quadratic bivalent programming problem, the (unconstrained) quadratic 0-1 programming problem, or the (unconstrained) quadratic (pseudo-) boolean programming problem (Beasley, 1998; Merz & Katayama, 2004).

The UBQP is known to be NP-hard (Garey & Johnson, 1979). It is a unified model for a variety of combinatorial optimization problems (Kochenberger, Glover, Alidaee, & Rego, 2004). The model (1), sometimes with some constraints added (which, however, can be reformulated as the model (1) (Kochenberger et al., 2004)), is of considerable practical significance. Numerous hard problems in many diverse areas have been formulated as the UBQP, including VLSI design, manufacturing, computer aided design, statistical mechanics, economics and finance, traffic management, molecular conformation, reliability theory and statistics, and so on. Moreover, many graph problems can be converted to the UBQP including the maximum clique problem, maximum cut problem, maximum vertex packing problem, minimum covering problem, and graph coloring problem (Kochenberger et al., 2004). Algorithms for the UBQP can be utilized to solve these problems as well. Therefore, solution approaches for the UBQP have been an active research area for many years due to the computational challenge and the vast potential applications (Beasley, 1998; Garey & Johnson, 1979; Kochenberger et al., 2004).

Several exact methods have been developed for the UBQP in the literature (Helmberg & Rendl, 1998; Palubeckis, 1995). Since however, the UBQP is known to be NP-hard (Garey & Johnson, 1979), large problems are proved to be intractable for these exact approaches. Several heuristic algorithms, local search (LS) methods, were proposed for the UBQP. These LS heuristic methods include one-pass LS (Boros, Hammer, & Tavares, 2007; Glover, Alidaee, Rego, & Kochenberger, 2002), 1-opt and k-opt LS methods (Merz & Freisleben, 2002).

Different kinds of metaheuristic algorithms were also proposed recently to find good solutions for the large UBQP problems. In 1998, Glover, Kochenberger, & Alidaee (1998) proposed a tabu search (TS) algorithm based on a flexible memory

E-mail address: wjiahai@hotmail.com

0957-4174/$ - see front matter © 2010 Elsevier Ltd. All rights reserved.
doi:10.1016/j.eswa.2010.02.032

system for UBQP instances with up to 500 variables. Beasley (1998) proposed another TS and a simulated annealing (SA), and tested the proposed TS and SA on several series of instances of size up to 2500 variables. Alkhamis, Hasab, & Ahmed (1998) also proposed a SA for the UBQP. Lodi, Allemand, & Liebling (1999) proposed an evolutionary algorithm (EA) for the small and medium size UBQP. Merz & Freisleben (1999) proposed a hybrid genetic algorithm incorporating 1-opt or *k*-opt LS and found new best solutions for several large Beasley problems. This hybrid genetic algorithm can also be called memetic algorithm (MA). Hasab, Alkhamis, & Ali (2000) presented a comparison among SA, GA and TS for the UBQP. Amini, Alidaee, & Kochemberger (1999) proposed a scatter search (SS). Katayama & Narihisa (2001) proposed a SA based on the 1-opt LS and multiple annealing processes. In 2004, Merz & Katayama (2004) proposed a very powerful MA with an innovative variation operator and a randomized *k*-opt LS. Palubeckis (2004) proposed five different multi-start tabu search (MST1-5) algorithms for UBQP instances with up to 6000 variables. Recently, Palubeckis (2006) proposed an iterated tabu search (ITS) for UBQP instances with up to 7000 variables.

When designing a heuristic algorithm for combinatorial optimization, it is preferable that it be simple, both conceptually and in practice. Further, it should be effective, and if possible, general purpose. One possible and very promising approach to combinatorial optimization problems is to apply Hopfield neural networks (HNNs) (Hopfield, 1982, 1984; Hopfield & Tank, 1985). For solving combinatorial optimization by the HNN, the desired objective function of the combinatorial optimization problem being optimized firstly is formulated as the energy function of the network, and then a neural updating rule is employed to guide the time evolution of the states of the network as it automatically seeks a minimum energy state of the energy function. Therefore, the HNN is a simple, effective and general-purpose heuristic algorithm for combinatorial optimization. Furthermore, the HNN is intrinsically easy to parallelize, especially on SIMD parallel computers (Blas, Jagota, & Hughey, 2005). In general, the HNN has two versions: continuous HNN (Hopfield, 1984) and discrete HNN (DHNN) (Hopfield, 1982). For the continuous HNN, neurons have continuous values within (0,1), and for the DHNN the neurons have only binary values, 0 or 1. Takefuji, Funabiki, and their co-workers (Funabiki, Takefuji, & Lee, 1993; Funabiki & Kitamichi, 1999; Funabiki & Nishikawa, 1997a, 1997b) 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 updates required to converge to a local minimum and the speed of executing each iteration.

The motivations of this research stem from two aspects. From previous review, we can find that different kinds of metaheuristic algorithms were proposed for the UBQP. However, previous studies using neural network (NN) for the UBQP are surprisingly scarce. This observation motivates our effort to propose a DHNN for the UBQP in this paper. As the past research has expressed (Funabiki et al., 1993; Funabiki & Kitamichi, 1999; Funabiki & Nishikawa, 1997a, 1997b), however, the discrete form of the solution search can easily bring on the problem of the local minimum convergence. The NN 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 are proposed such as multi-start DHNN (He, Sykora, & Makinen, 2006; He & Sykora, 2006) and DHNN with random flips (Smitha, Abramsonb, & Duke, 2003). How to help the DHNN escape from local minima and further improve the performance of network is a key issue in the HNN field, which motivates our effort to propose new idea or method to deal with the local minima problem of the network.

In the last decade, more and more researchers tried to overcome the drawbacks of usual recombination operators of evolutionary computation algorithms. Therefore, estimation of distribution algorithms (EDAs) (Baluja, 1994; Harik, Lobo, & Goldberg, 1999; Larranaga & Lozano, 2001; Muhlenbein & Paaß, 1996; Pelikan, Goldberg, & Lobo, 2002; Zhang, Sun, & Tsang, 2004, 2005; Zhang, Sun, Xiao, & Tsang, 2007; Zhang & Sun, 2006) have been developed. These algorithms, which have a theoretical foundation in probability theory, are also based on populations that evolve as the search progresses. EDAs use probabilistic modeling of promising solutions to estimate a distribution over the search space, which is then used to produce the next generation by sampling the search space according to the estimated distribution. After every iteration, the distribution is re-estimated.

In order to overcome the local minima problem of the DHNN, the idea of EDA is combined with DHNN and a DHNN–EDA is proposed for the UBQP 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, which is in a promising area characterized by a probability model. Thus, the DHNN–EDA can escape from local minima and further search better results. The performance of the DHNN–EDA is tested and evaluated by simulating a large number of benchmark instances. Firstly, the DHNN–EDA is compared with other improved DHNN algorithms. Simulation results show the superior performance of the DHNN–EDA. Secondly, the DHNN–EDA is compared with metaheuristic algorithms. Simulation results show that the DHNN–EDA is better than or competitive with other metaheuristic algorithms such as TS, SA, SS and MA. Finally, we mention the transiently chaotic neural network (TCNN) (Chen & Aihara, 1995) and noisy chaotic neural network (NCNN) (Wang, Li, Tian, & Fu, 2004, 2008) 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 DHNN–EDA. Further, we discuss the DHNN–EDA in the multi-start algorithm framework and clarify the similarities and differences between the DHNN–EDA and other metaheuristic algorithms, which suggests or enlightens several ideas to further improve the performance of the DHNN–EDA.

The contributions of this paper thus consist in several aspects: Firstly, since previous studies using NN for the UBQP are surprisingly scarce, a novel NN based algorithm, named DHNN–EDA, is proposed for the UBQP and very competitive results are obtained, which is the contribution to the available literature on the UBQP. Though the DHNN–EDA is not the best algorithm for the UBQP so far, it can be improved further in the future, and now can be seen as the first step or meaningful attempt of adopting the NN method to deal with the UBQP. Secondly, the probabilistic modeling/EDA idea is applied to HNN training and helps the network escape from local minima. Thus, a novel NN model is proposed and can be applied to other combinatorial optimization problems, which is the contribution to the HNN literature. Finally, a comprehensive experimental comparison of the DHNN–EDA with other methods is also provided.

The remaining sections of this paper are organized as follows. In Section 2, we propose a DHNN for the UBQP. In Section 3, we propose a DHNN combined with EDA (DHNN–EDA) for the UBQP. In Section 4, benchmark data sets are used to evaluate the proposed DHNN–EDA. Last section gives several research lines in the future and concludes this paper.

2. DHNN for UBQP

The UBQP with a symmetric rational $n \times n$ matrix $Q = (q_{ij})$ can be mapped onto the DHNN with n neurons. The ith neuron has

input u_i and output v_i. According to objective function Eq. (2), the energy function of the UBQP can be written as:

$$E = -\sum_{i=1}^{n}\sum_{j=1,j\neq i}^{n} q_{ij} v_i v_j - \sum_{i=1}^{n} q_{ii} v_i. \quad (3)$$

According to the update rule of the DHNN (Hopfield, 1982), the inputs of the neurons are computed by the following motion equation:

$$u_i = -\frac{\partial E}{\partial v_i} = \sum_{j=1,j\neq i}^{n} q_{ij} v_j + q_{ii}. \quad (4)$$

The binary input/output function of the ith neuron is given by:

$$v_i(t+1) = \begin{cases} 1 & \text{if } u_i(t) > 0 \\ 0 & \text{otherwise} \end{cases} \quad (5)$$

At the beginning, the network is randomly given an initial input $u_i \in (-1,1)$. Correspondingly, the neuron states of the network are randomly initialized with 0 or 1. The DHNN can thus be summarized as follows:

DHNN for UBQP

initialize **u** (randomly around zero);
calculate **v** using Eq. (5);
$t = 0$;
repeat
  copy the state array of neurons **v** to **lastv**;
  for $i = 1$ to n **do**/*HNN descent procedure*/
  compute $u_i(t+1)$ with motion Eq. (4);
  update v_i using Eq. (5);
end for
$t = t + 1$;
until ($t >$ *iterations* or stability criteria is satisfied);

The neurons in the network are updated in a sequential update. The stability criterion is to see if the current state **v** is equal to last state of the neuron array **lastv**.

This DHNN can make significant advances in the earlier stages of optimization procedures but slow down significantly as they get close to the optimal solution. That is, the discrete form of the solution search can easily bring on the problem of the local minimum convergence. The NN in the local minimum cannot move to other states. In the next section, we will propose a DHNN combined with EDA for the UBQP in order to overcome the local minima problem of the DHNN.

3. DHNN combined with EDA for UBQP

This DHNN can make significant advances in the earlier stages of optimization procedures (that is, the energy function decreases rapidly and dramatically in the earlier stages), and improvements slow thereafter, until after several iterations, for example 10 iterations, a local minimum is encountered and thus the energy function no longer decreases (Smitha et al., 2003). The local minima 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 current 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, which is in a promising area characterized by a probability model. In the following subsections, we first briefly review the EDA, and then the DHNN–EDA is proposed. Finally, complexity analysis of the DHNN–EDA is given.

3.1. Estimation of Distribution Algorithm (EDA)

Estimation of Distribution Algorithm (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 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:

Framework of EDA

Pop = InitializePopulation() /*Initialization*/
while Stopping criteria are not satisfied **do** /*Main Loop*/
Pop_{sel} = Select(*Pop*)/*Selection*/
Pro*p* = *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. More comprehensive presentation of the EDA field can be found in Larranaga & Lozano (2001), Pelikan et al. (2002).

3.2. DHNN combined with EDA (DHNN–EDA)

The EDA is a novel optimization tool and primarily used in evolutionary algorithms which are population-based 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 generation of offspring in original EDA (Zhang et al., 2004, Zhang, Sun, & Tsang, 2005; Zhang et al., 2007; Zhang & Sun, 2006). Therefore, how to combine EDAs with other techniques represents an important research direction (Zhang et al., 2004). Recently, Zhang et al. (2004, 2005), Zhang et al. (2007), Zhang & Sun (2006) proposed several works on EDA hybrids for hard optimization problems. Especially, 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 (Zhang et al., 2004, 2005; Zhang et al., 2007; Zhang & Sun, 2006). Considering the local minima problem of the DHNN and these successes recently reported in effectively combining EDA and other techniques (Zhang et al., 2004, 2005; Zhang et al., 2007; Zhang & Sun, 2006), the EDA is introduced into the DHNN, which is a single point search method, to help the DHNN escape from local minima in this paper. The DHNN–EDA for the UBQP can thus be summarized as follows:

DHNN–EDA for UBQP

initialize **u** (randomly around zero);
calculate **v** using Eq. (5);
initialize best-so-far solution **gb** and probability model P;
for $k = 1$ to *descents* **do**
$t = 0$;
  repeat
    copy the state array of neurons **v** to **lastv**;
    for $i = 1$ to n **do**/*HNN descent procedure*/
    compute $u_i(t+1)$ with motion Eq. (4);
    update v_i using Eq. (5);
  end for
$t = t + 1$;
until ($t >$ *iterations* or stability criteria is satisfied);
update and keep track for the best-so-far solution **gb**;
update probability model using **v**;
perturb the current solution **v** using EDA mutation to generate a new starting point **v** for the HNN;
end for

The descents of HNN will visit a number of locally optimal solutions. Statistics 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 (Muhlenbein & Paaß, 1996), population-based incremental learning (PBIL) (Baluja, 1994), compact GA (Harik et al., 1999). 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 DHNN–EDA uses a probability vector $P = (p_1, \ldots, p_i, \ldots, p_n)$ 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 P, which characterizes distribution of promising solutions. It can be expected that offspring fall in or close to a promising area in the search space. The EDA mutation perturbs the current solution **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 $v_i(t+1) = 1$, otherwise set $v_i(t+1) = 0$;
  Otherwise, $v_i(t+1) = v_i(t)$ /*Copy from current solution **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 P randomly or directly copied from the current solution **v**, which is controlled or balanced by the parameter β. The larger β is, the more elements of the new starting point are sampled from the vector P. Since some elements of offspring are sample from the probability vector 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 afterwards.

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 to leave the current local minimum and to enable the HNN to find new, 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 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 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 randomly generated solutions.

In the DHNN–EDA, the probability vector P is initialized as $P = (0.5, \ldots, 0.5, \ldots, 0.5)$. 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 (Baluja, 1994):

$$p_i(t+1) = (1-\lambda)p_i(t) + \lambda v_i(t); \quad (6)$$

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 DHNN–EDA, current locally optimal solution **v** generated by the HNN descent procedure is selected, then **v** is used in the estimation stage in which the probability model P is updated according to the rule defined by Eq. (6). The sampling process is described in the EDA mutation operator. However, there are several differences between the DHNN–EDA and the most usual population-based EDA algorithms. Firstly, in the sampling process of the DHNN–EDA, the probability model is not sampled directly as in the pure or original EDA, but rather used as part of the EDA mutation. Secondly, although the DHNN–EDA uses a univariate probability model in the style of the PBIL, only one solution is sampled from the probability model and only one solution is used to update the model (that is, the population size is equal to 1). Thus, the DHNN–EDA is more akin to the EDA hybrid, the guided mutation (Zhang et al., 2004, 2005; Zhang et al., 2007; Zhang & Sun, 2006). In contrast to that the guided mutation acts on the best-so-far solution **gb**, the EDA in the DHNN–EDA mutates the current solution **v** in order to increase the diversification level and thus exploration ability in the search process. Furthermore, the main novelty of the DHNN–EDA consists in applying the probabilistic modeling/EDA idea to train HNN and thus help the network escape from local minima.

Guided by a probability model which characterizes the distribution of promising solutions in the search space, the EDA mutation mutates the current solution 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.

3.3. Complexity analysis

The DHNN–EDA 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 invocation and then can be seen as a stopping criterion. One updating iteration is defined as one complete update of all n neurons according to Eqs. (4) and (5). The maximum iteration number of the DHNN–EDA is *descents* $\times$ *iterations*.

For the UBQP with a symmetric rational $n \times n$ matrix $Q = (q_{ij})$, the HNN uses n neurons. According to Eq. (4), the calculation in each neuron takes time $O(n)$. Therefore, a sequential traversal of all n neurons takes time $O(n^2)$ in the DHNN–EDA. Updating of

the probability model takes time $O(n)$. Therefore, the whole complexity of the DHNN–EDA is $O(n^2)$.

4. Simulation results and discussion

In order to assess the performance of the DHNN–EDA, simulations were implemented in C on a PC (Pentium 4 2.80 GHz). In this section we describe the benchmark datasets, the parameter setting, and present the results of the DHNN–EDA on the benchmark datasets. We also compare the results with those DHNN algorithms with different modifications to demonstrate the effect of the EDA perturbation, and compare the results with other metaheuristics such as SA, TS, GA, and MA.

4.1. Benchmark datasets

In this paper, we test the DHNN–EDA on 56 benchmark problems which are also used in numerous other studies of the UBQP (Beasley, 1998; Merz & Katayama, 2004; Palubeckis, 2006). These benchmark problems include two groups. The first group has 35 problems and the second group has 21 randomly generated problems of larger size and higher density. The size of these benchmark problems ranges from 500 (small scale) to 7000 (large scale).

The basic parameters and the best-known solutions of first group including 35 problems are shown in Table 1. In Table 1, density of the problems means the ratio of the number of nonzero coefficients of quadratic terms in Eq. (2). Minimum and maximum of the coefficients of the linear and quadratic terms are also provided. The glov500 set in the first group is studied by Glover et al. (1998). The glov500 set has five instances of size $n = 500$ with a matrix density varying between 0.1 and 1.0. The second set kb-g1000 is kindly provided by Kochenberger and has been used to test the performance of SS (Amini et al., 1999). The kb-g1000 set has 10 instances of size $n = 1000$ with a matrix density varying between 0.1 and 1.0. The bqp1000 ($n = 1000$) and bqp2500 ($n = 2500$) were first studied by Beasley (1998) and each set has 10 instances with a matrix density 0.1.

The glove500, bqp1000 and bqp2500 set in Table 1 were taken from the OR-Library website (Beasley, 2008), and the bqp2500 set is the largest UBQP test set in the OR-Library. The kb-g1000 set was kindly provided by Merz & Katayama (2004). There are several small-scale test instances ($n < 500$ with varying densities) contained in the OR-Library website. These instances are not considered in our experiments, since they are solved easily by heuristics in very short time and thus no good comparisons among algorithms can be provided. Note that the UBQP is formulated as a minimum problem in Glover et al. (2002), while the UBQP is formulated as a maximum problem in this paper just like in Beasley (1998), Merz & Katayama (2004), Kochenberger et al. (2004), Boros et al. (2007).

In order to further show the performance of the DHNN–EDA, the DHNN–EDA is tested on the second group including 21 randomly generated problems of larger size and higher density which is first produced and used in the Palubeckis (2006). These 21 problems have density from 50% to 100% and vary in size from 3000 to 7000 variables. All nonzero coefficients of the objective function are drawn uniformly at random from the interval $[-100, 100]$. The basic parameters and the best-known solutions of 21 problems are shown in Table 2. The sources of the generator and input files

Table 1
Main characteristics of 35 benchmark test problems in the first group for UBQP.

Set	No.	n	Density	Linear coef. q_{ii}	Quadr. coef. q_{ij}	Best known solution
glov500	1	500	0.1	[−75,75]	[−50,50]	61,194
	2	500	0.25	[−75,75]	[−50,50]	100,161
	3	500	0.5	[−75,75]	[−50,50]	138,035
	4	500	0.75	[−75,75]	[−50,50]	172,771
	5	500	1	[−75,75]	[−50,50]	190,507
kb-g1000	1	1000	0.1	[−75,75]	[−50,50]	131,456
	2	1000	0.2	[−75,75]	[−50,50]	172,788
	3	1000	0.3	[−75,75]	[−50,50]	192,565
	4	1000	0.4	[−75,75]	[−50,50]	215,679
	5	1000	0.5	[−75,75]	[−50,50]	242,367
	6	1000	0.6	[−75,75]	[−50,50]	243,293
	7	1000	0.7	[−75,75]	[−50,50]	253,590
	8	1000	0.8	[−75,75]	[−50,50]	264,268
	9	1000	0.9	[−75,75]	[−50,50]	262,656
	10	1000	1	[−75,75]	[−50,50]	274,375
bqp1000	1	1000	0.1	[−100,100]	[−100,100]	371,438
	2	1000	0.1	[−100,100]	[−100,100]	354,932
	3	1000	0.1	[−100,100]	[−100,100]	371,236
	4	1000	0.1	[−100,100]	[−100,100]	370,675
	5	1000	0.1	[−100,100]	[−100,100]	352,760
	6	1000	0.1	[−100,100]	[−100,100]	359,629
	7	1000	0.1	[−100,100]	[−100,100]	371,193
	8	1000	0.1	[−100,100]	[−100,100]	351,994
	9	1000	0.1	[−100,100]	[−100,100]	349,337
	10	1000	0.1	[−100,100]	[−100,100]	351,415
bqp2500	1	2500	0.1	[−100,100]	[−100,100]	1,515,944
	2	2500	0.1	[−100,100]	[−100,100]	1,471,392
	3	2500	0.1	[−100,100]	[−100,100]	1,414,192
	4	2500	0.1	[−100,100]	[−100,100]	1,507,701
	5	2500	0.1	[−100,100]	[−100,100]	1,491,816
	6	2500	0.1	[−100,100]	[−100,100]	1,469,162
	7	2500	0.1	[−100,100]	[−100,100]	1,479,040
	8	2500	0.1	[−100,100]	[−100,100]	1,484,199
	9	2500	0.1	[−100,100]	[−100,100]	1,482,413
	10	2500	0.1	[−100,100]	[−100,100]	1,483,355

Table 2
Main characteristics of 21 benchmark test problems with larger size and higher density in the second group for UBQP.

Set	No.	n	Density	Linear coef. q_{ii} and quadr. coef. q_{ij}	Best known solution
p3000	1	3000	0.5	[−100,100]	3,931,583
	2	3000	0.8		5,193,073
	3	3000	0.8		5,111,533
	4	3000	1		5,761,822
	5	3000	1		5,675,625
p4000	1	4000	0.5		6,181,830
	2	4000	0.8		7,801,355
	3	4000	0.8		7,741,685
	4	4000	1		8,711,822
	5	4000	1		8,908,979
p5000	1	5000	0.5		8,559,355
	2	5000	0.8		10,836,019
	3	5000	0.8		10,489,137
	4	5000	1		12,252,318
	5	5000	1		12,731,803
p6000	1	6000	0.5		11,384,976
	2	6000	0.8		14,333,855
	3	6000	1		16,132,915
p7000	1	7000	0.5		14,478,676
	2	7000	0.8		18,249,948
	3	7000	1		20,446,407

to replicate these problem instances can be found at http//:www.softer.ktu.lt/~gintaras/ubqop_its.html.

4.2. Parameter setting

The DHNN–EDA has two main parameters, 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 λ is, the greater the contribution of current local minimum is.

To assess the effects of these parameters, the DHNN–EDA is tested as a pre-experimental tuning on a representative subset of problems with different size and density from Tables 1 and 2. These selected problems include glov500-5, kb-gb1000-10, bqp1000-1, bqp2500-1, p3000-1, p4000-2, p5000-3, p5000-4, p6000-3 and p7000-3.

Firstly, 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 results using these parameter combinations. This suggests that relatively small mutation strength and relatively small contribution of the current local minimum is adequate for the UBQP.

Secondly, 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. The problem for too strong mutation strength and too big contribution of the current solution is that the resulting algorithm is very similar to repeating the HNN search from randomly generated or initialized starting point like the multi-start HNN (He et al., 2006). On the other hand, since PBIL described by Eq. (6) is an incremental learning method, the learning rate should be set to a very small value, or else the probability model will "forget" the old information extracted from the historical local minima quickly.

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 shown in Tables 1 and 2 in the whole simulation. Certainly, the parameters are chosen experimentally, and 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 DHNN for the UBQP works hardest and most productively, measured by a rapidly decreasing energy, during the first eight iterations. Therefore, the maximum iteration number of each descent in the DHNN–EDA is set to 10 iterations, and each test run is set to invoke 500 times HNN search, that is, *descents* = 500. The maximum iteration number of the DHNN–EDA is 10 * 500 = 5000. This is a trade-off between solution quality and computation time. If given more computation time, the DHNN–EDA can implement or invoke more times HNN search and therefore, can be expected to produce better results.

4.3. Comparison with HNN algorithms

In this section, we first describe two modified DHNN algorithms proposed for combinatorial optimization problems recently (He et al., 2006; He & Sykora, 2006; Smitha et al., 2003), and then we compare the DHNN–EDA with them to show how the EDA process can improve the performance of the DHNN.

He et al. (2006) proposed an improved HNN which repeats the random initial solution construction and HNN search a number of times and returns the best solution. Therefore, the improved HNN is called multi-start DHNN algorithm in this paper. He et al. applied the multi-start DHNN to two-page crossing number problem (TPCNP) and outerplanar drawing problem and obtained better results than other NNs (He et al., 2006; He & Sykora, 2006). The structure of the multi-start DHNN algorithm is as follows:

Multi-Start DHNN for UBQP

```
initialize the best-so-far solution gb;
for k = 1 to descents do
  initialize u (randomly around zero);
  calculate v using Eq. (5);
  t = 0;
  repeat
    copy the state array of neurons v to lastv;
    for i = 1 to n do/*HNN descent procedure*/
      compute u_i(t + 1) with motion Eq. (4);
      update v_i using Eq. (5);
    end for
t = t + 1;
until (t > iterations or stability criteria is satisfied);
update and keep track for the best-so-far solution gb;
end for
```

In the multi-start DHNN, HNN descent procedure is restarted with different initial states of the neurons for several times, and 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.

Smitha et al. (2003) 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 in this paper. The simulation results on timetabling problems show that the DHNN with random flips is comparable to or better than the SA, TS and greedy search methods (Smitha et al., 2003). 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 **gb**, to the original DHNN with random flips. DHNN with random flips can thus be summarized as follows:

DHNN with random flips for UBQP

```
initialize u (randomly around zero);
calculate v using Eq. (5);
initialize best-so-far solution gb;
for k = 1 to descents do
  t = 0;
  repeat
    copy the state array of neurons v to lastv;
    for i = 1 to n do/*HNN descent procedure*/
      compute u_i(t + 1) with motion Eq. (4);
      update v_i using Eq. (5);
    end for
  t = t + 1;
until (t > iterations or stability criteria is satisfied);
update and keep track for the best-so-far solution gb;
randomly choose a neuron, and flip according to threshold;
end for
```

In the DHNN 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 say 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 above 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 Smitha et al. (2003). Note that the DHNN with random flips in this paper uses the sequential updating, while the original DHNN with random flips uses the parallel updating.

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

Table 3 shows the best, average and standard deviation values produced by the multi-start DHNN, DHNN with random flips, and the DHNN–EDA, respectively, for the 35 test problem instances over 30 independent runs. The best-known values of these test problem instances are also shown in Table 3. The last two rows show the average solution values and the average percent deviation of the solutions found by each algorithm from the best-known values shown in the second column of Table 3. From Table 3, we find that the DHNN with random flips gets slightly better best results than the multi-start DHNN. But the DHNN with random flips is worse than the multi-start DHNN in terms of average solutions and standard deviation. It is apparent that the DHNN–EDA performs better than the multi-start DHNN and DHNN with random flips in all instances in terms of best, average and standard deviation values. The average results of the DHNN–EDA are even better than the best results of the multi-start DHNN and DHNN with random flips.

The multi-start DHNN and DHNN with random flips find only one best-known value for small size instance, respectively. The DHNN–EDA can obtain the best-known value in 27 out of 35 instances. Bold figures indicate the best-known value obtained by all the algorithms. The average percent deviation of the best solutions found by the multi-start DHNN, DHNN with random flips, and the DHNN–EDA from the best-known values for 35 instances are 0.25858%, 0.25051%, and 0.00580%, respectively. The average percent deviation of the average solutions found by the multi-start DHNN, DHNN with random flips, and the DHNN–EDA from the best known values for 35 instances are 0.47952%, 0.66452%, and 0.05096%, respectively. The average standard deviation of the multi-start DHNN, DHNN with random flips, and the DHNN–EDA for 35 instances are 597.48, 1831.32 and 138.57, respectively. The low standard deviation of the DHNN–EDA reveals that the DHNN–EDA guided by the EDA model has high stability.

In the multi-start DHNN, random initialization is a random sampling mechanism in the search space. That is, the starting solutions are generated completely randomly and thus each HNN search is independently invoked. It does not take into consideration any information from the previous HNN search and can be seen as a memory-less method. If it is lucky enough to start from a good initial point, the multi-start HNN can converge to a good solution. In contrast, the DHNN with random flips and the DHNN–EDA all alternate between calls to HNN search and solution perturbation procedures. Each time the solution perturbation procedure is applied to the solution coming from the previous HNN search, and thus the HNN descents are not independently invoked. Therefore, the search history of previous HNN search is likely utilized by the subsequent HNN search to help the network escape from local minima. However, there is difference between the DHNN with random flips and the DHNN–EDA. The DHNN with random flips uses random perturbation or mutation on the just last solution to help the next HNN escape from local minima, and thus does not explicitly utilize global statistical information in the previous search. Further, which neuron state is chosen to be flipped is decided by a completely random way, and therefore, it is somewhat "blind", which leads to high standard deviation. The high standard deviation reveals that the method has low stability. In the DHNN–EDA, a probability vector is built to characterize the distribution of promising solutions in the search space. The DHNN–EDA is guided by the global search information in EDA model,

Table 3
Comparison of the results obtained by the multi-start DHNN, DHNN with random flipping perturbation, and the DHNN–EDA for the 35 test problem instances.

Problem	Best known solution	Multi-start DHNN			DHNN with random flips			DHNN–EDA		
		Best	Av.	Std.	Best	Av.	Std.	Best	Av.	Std.
glov500-1	61,194	61,025	60,686.53	159.96	**61,194**	60,846.80	257.06	**61,194**	61,192.97	2.54
2	100,161	**100,161**	99,673.07	223.16	100,158	99,619.60	386.98	**100,161**	100,161.00	0
3	138,035	137,735	137,126.13	349.25	137,946	137,217.40	587.23	**138,035**	138,026.70	17.46
4	172,771	172,685	172,121.73	263.00	172,618	172,002.43	455.94	**172,771**	172,710.40	61.28
5	190,507	190,502	189,771.73	405.59	190,502	189,264.77	1022.71	**190,507**	190,505.83	2.11
kb-g1000-1	131,456	130,678	130,240.93	247.31	131,140	130,328.33	384.62	**131,456**	131,385.93	34.37
2	172,788	172,016	171,029.17	468.37	172,020	170,447.17	1194.10	**172,788**	172,731.37	55.81
3	192,565	191,503	189,584.43	754.74	191,258	187,528.33	2289.42	**192,565**	192,565.00	0
4	215,679	214,213	212,900.73	615.28	214,519	211,720.97	1557.20	**215,679**	215,064.20	131.23
5	242,367	242,345	240,767.83	660.53	241,580	239,330.10	1580.52	**242,367**	242,365.50	1.50
6	243,293	241,459	239,523.10	779.10	241,501	237,477.46	2737.43	**243,293**	243,229.03	59.39
7	253,590	250,694	249,050.77	814.26	252,135	246,998.57	2768.30	**253,590**	253,120.87	291.13
8	264,268	262,489	261,137.43	628.70	262,997	259,625.50	2604.30	**264,268**	264,043.27	99.39
9	262,658	259,887	258,383.60	820.00	259,227	254,363.90	3086.87	**262,658**	262,391.57	172.84
10	274,375	273,271	270,041.43	1023.72	270,435	266,117.97	2492.84	**274,375**	274,093.60	251.44
bqp1000-1	371,438	370,905	370,342.43	279.22	371,134	370,463.0	481.69	**371,438**	371,310.83	91.27
2	354,932	354,128	353,080.17	495.50	354,358	352,534.23	1008.37	**354,932**	354,837.67	86.20
3	371,236	370,700	369,243.97	558.52	370,988	369,153.60	1123.58	**371,236**	371,103.87	80.32
4	370,675	370,085	369,012.90	468.31	370,303	368,636.43	11,490.88	**370,675**	370,534.87	68.52
5	352,760	351,855	350,859.87	449.96	352,527	350,856.80	1088.89	**352,760**	352,631.17	76.38
6	359,629	358,884	358,083.30	338.17	359,239	358,100.83	762.67	**359,629**	359,419.53	103.43
7	371,193	369,952	369,224.90	307.94	370,215	369,252.83	677.29	**371,193**	370,874.10	140.31
8	351,994	350,725	349,522.43	541.15	351,253	349,452.77	903.27	**351,994**	351,800.20	92.49
9	349,337	347,951	347,126.97	397.98	348,768	346,962.17	1093.68	**349,337**	349,140.97	179.46
10	351,415	350,454	349,802.50	381.65	350,827	349,846.63	815.03	**351,415**	351,144.80	205.43
bqp2500-1	1,515,944	1,512,359	1,509,857.43	1162.69	1,512,877	1,508,995.13	1841.37	1,515,807	1,515,518.30	148.33
2	1,471,392	1,467,065	1,465,777.67	633.02	1,467,898	1,463,879.27	2621.60	1,471,106	1,470,316.07	344.69
3	1,414,192	1,411,094	1,408,718.37	935.35	1,410,514	1,406,477.67	2136.87	**1,414,192**	1,413,492.13	247.17
4	1,507,701	1,504,718	1,503,179.57	719.75	1,505,985	1,501,354.77	2311.30	**1,507,701**	1,507,324.60	254.74
5	1,491,816	1,490,066	1,487,967.53	1005.74	1,490,053	1,485,576.20	2356.85	1,491,770	1,491,473.60	148.17
6	1,469,162	1,466,437	1,464,173.67	760.20	1,466,600	1,462,834.53	2090.28	1,468,977	1,468,179.57	333.22
7	1,479,040	1,475,332	1,473,347.90	926.94	1,474,071	1,470,887.53	2142.05	1,478,844	1,478,204.77	284.12
8	1,484,199	1,481,706	1,480,690.33	446.02	1,482,801	1,479,910.87	1598.04	1,484,159	1,483,613.47	234.35
9	1,482,413	1,479,985	1,477,988.33	838.29	1,478,055	1,474,911.10	1956.88	1,482,302	1,482,000.93	161.98
10	1,483,355	1,479,336	1,477,258.40	1052.31	1,478,427	1,474,881.37	2669.97	1,483,122	1,482,155.33	388.79
Av.	609,129.37	607,554.29	606,208.49	597.48	607,603.51	605,081.63	1831.32	609,094.11	608,818.97	138.57
Deviation%	–	0.25858	0.47952	–	0.25051	0.66452	–	0.00580	0.05096	–

and therefore, can search better solution in the promising region, which is the main reason that the DHNN–EDA can obtain better results than the multi-start DHNN and the DHNN with random flips. On the other hand, the probability vector extracted from the previous search history can be seen as a long term memory. The use of memory is nowadays recognized as one of the fundamental elements of a powerful heuristic algorithm.

Table 4 shows the computation time of the three algorithms in second. Only one neuron in the DHNN with random flips is chose to be flipped, therefore, the network maybe return to the old local minima after very few iterations. The DHNN–EDA mutates the current local minimum by the EDA mutation and it also can converge to next local minimum quickly. Therefore, the DHNN–EDA is much faster than the multi-start HNN starting from a randomly generated solution. Hence, for the same given computation time, the DHNN–EDA can invoke more HNN descents than multi-start HNN. The DHNN–EDA has good balance between the computation time and solution quality.

In summary, the probabilistic modeling/EDA idea is applied to HNN training and helps the network escape from local minima. Thus, the performance of the DHNN is greatly improved due to the *EDA process*, which is our main contribution in this paper.

4.4. Comparison with metaheuristic algorithms

In order to further show the effectiveness of the DHNN–EDA, we now present a comparative performance against some metaheuristic algorithms including the TS proposed by Beasley (TS-B), the SA proposed by Beasley (SA-B) (Beasley, 1998), the SS proposed by Amini (SS-A) (Amini et al., 1999), the EA proposed by Lodi et al. (1999), MA proposed by and Merz and Freisleben (MA-MF) (Merz & Freisleben, 1999) and MA proposed by Merz and Ktayama (MA-MK) (Merz & Katayama, 2004). These algorithms have been proposed as one of the best heuristic approaches to the UBQP in those days.

Table 5 displays the name of the instances, the best-known solution value, the results produced by the metaheuristic algorithms and the DHNN–EDA. The results of the competitors are directly adopted from the original papers of the competitors. For bqp1000 and bqp2500 instances, the last two rows show the average solution values and the average percent deviation of the solutions found by each algorithm from the best known values.

Only the best solution value obtained is provided for the EA for glov500 instances. The EA and the DHNN–EDA also can get the best-known values for all glov500 instances.

For the SS-A, only the best solution value for the sets glov500 and kb-g1000 are available. The DHNN–EDA and SS-A also can get the best known values for all glov500 and kb-g1000 instances.

For the TS-B and SA-B, only the best solution value for each instance of set glov500 and the problem sets bqp1000 and bqp2500 are available. The two sets of bqp1000 and bqp2500 are first provided in Beasley (1998) as new benchmark instances contained in the ORLIB (Beasley, 2008). In those days, the values of the solutions obtained by these algorithms were reported as the best-known solutions for each instance of the two sets. Comparing with SA-B, the DHNN–EDA can obtain better solution for all instances. Further, the average results of the DHNN–EDA are better than the best results of SA-B in the whole. Compare with the TA-B, the DHNN–EDA also can obtain better solutions for all instances except that the DHNN–EDA get the same solutions for glov500-1 and glov500-4 instances, and get worse solution for bqp2500-5, bqp2500-8 and bqp2500-9 instances. Further, the average results of the DHNN–EDA are better than the best results of the TA-B for bqp2500 instances in the whole.

Table 4
Comparison of average computational time of the multi-start DHNN, DHNN with random flipping perturbation, and the DHNN–EDA for the 35 test problem instances.

Problems	Multi-start DHNN	DHNN with random flips	DHNN–EDA
glov500	14.32	3.60	7.38
kb-g1000	66.94	13.93	34.67
bqp1000	57.53	13.83	34.10
bqp2500	511.24	107.94	276.95

For the MA-MF, the average solution values are provided. Although the DHNN–EDA is slightly worse than the MA-MF for glov500 and bqp1000 instances in terms of average solution value, the DHNN–EDA can obtain better results for all bqp2500 instances except for bqp2500-4 and bqp2500-5 instances.

For the MA-MK, only the average solution value for each instance is given because the MA-MK can obtain all the best-known values for the 35 instances in 30 independent runs. It is not surprising that the DHNN–EDA is outperformed by the MA-MK. The MA-MK includes a randomized greedy heuristic to create the initial population, a highly effective randomized k-opt LS to obtain a population of locally optimum solutions, and a new recombination-based variation operator, called innovative variation, to produce good starting points for the LS. The initialization, recombination-based variation operator, and strong LS are all tailored specifically for the UBQP by taking the properties of the fitness landscape of the UBQP into account. In contrast to the MA-MK, the DHNN–EDA does not include any specifically design or consideration for the UBQP.

4.5. Comparison of results for larger size and higher density instances

Though the bqp2500 instances are the largest instances in the OR-Library, Palubeckis (2006) pointed out that those problem instances are not sufficiently strong for state-of-the-art algorithms for the UBQP. Therefore, Palubeckis proposed five different MST algorithms (MST1–MST5) (Palubeckis, 2004) and an ITS algorithm (Palubeckis, 2006) and tested these TS based algorithms on a set of randomly generated problems of larger size and higher density. These 21 generated problems have density from 50% to 100% and vary in size from 3000 to 7000 variables. In order to further show the performance of the DHNN–EDA, the DHNN–EDA is also tested on these larger size problems and compared with other algorithms.

The ITS and MST2 are the most powerful algorithms for the UBQP so far, therefore, the ITS and MST2 can be used as the benchmark algorithms to evaluate the performance of the DHNN–EDA.

The main ingredients of the ITS include solution perturbation procedures, get start point (GSP), for construction of a starting solution, and simple tabu search (STS) for iterative improvement of this solution. The ITS repeatedly invokes this two procedures. In the ITS for the UBQP, the STS procedure is an adaptation of a simple TS implementation proposed by Beasley (1998). The STS consists of a best improvement LS and a short term memory to escape from local minima and to 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 TS procedure described by Beasley (1998) is $O(n^2)$. But a computational trick, mapping of the current solution to the zero vector, is applied in the STS for the UBQP. Thus, the ITS in fact work with a transformed instances of Eqs. (1) and (2). The trick of transformation or modification designed or tailored specifically for the UBQP, allows to hasten exploration of the neighborhood of the current solution and reduces the computation time significantly. Therefore, the complexity of TS procedure in the ITS decreases to $O(n)$. The computation trick also plays the crucial role in the solution perturbation procedure. The perturbation procedure is to select variables that must undergo a flipping operation. These variables are randomly selected from a candidate list. The

Table 5
Comparison of the results obtained by the five metaheuristic methods and the DHNN–EDA for the 35 test problem instances.

Problem	Best known solution	MA-MK	MA-MF	TS-B	SA-B	SS-A	EA	DHNN–EDA	
		Av.	Av.	Best	Best	Best	Best	Best	Av.
glov500-1	61,194	**61,194**	**61,194**	**61,194**	61,183	**61,194**	**61,194**	**61,194**	61,192.97
2	100,161	**100,161**	**100,161**	100,158	100,158	**100,161**	**100,161**	**100,161**	100,161.00
3	138,035	**138,035**	**138,035**	138,021	**138,035**	**138,035**	**138,035**	**138,035**	138,026.70
4	172,771	**172,771**	-	**172,771**	172,150	**172,771**	**172,771**	**172,771**	172,710.40
5	190,507	**190,507**	–	190,502	190,498	**190,507**	**190,507**	**190,507**	190,505.83
kb-g1000-1	131,456	**131,456**	–	–	–	**131,456**	–	**131,456**	131,385.93
2	172,788	**172,788**	–	–	–	**172,788**	–	**172,788**	172,731.37
3	192,565	**192,565**	–	–	–	**192,565**	–	**192,565**	192,565.00
4	215,679	**215,679**	–	–	–	**215,679**	–	**215,679**	215,064.20
5	242,367	**242,367**	–	–	–	**242,367**	–	**242,367**	242,365.50
6	243,293	**243,293**	–	–	–	**243,293**	–	**243,293**	243,229.03
7	253,590	**253,590**	–	–	–	**253,590**	–	**253,590**	253,120.87
8	264,268	**264,268**	–	–	–	**264,268**	–	**264,268**	264,043.27
9	262,658	262,618	–	–	–	**262,658**	–	**262,658**	262,391.57
10	274,375	274,335.4	–	–	–	**274,375**	–	**274,375**	274,093.60
bqp1000-1	371,438	**371,438**	371,304.1	**371,438**	371,124	–	–	**371,438**	371,310.83
2	354,932	**354,932**	**354,862.3**	**354,932**	354,637	–	–	**354,932**	354,837.67
3	371,236	**371,236**	**371,233.8**	371,073	371,226	–	–	**371,236**	371,103.87
4	370,675	**370,675**	370,506.0	370,560	370,265	–	–	**370,675**	370,534.87
5	352,760	**352,760**	**352,687.6**	352,736	352,297	–	–	**352,760**	352,631.17
6	359,629	**359,629**	**359,487.8**	359,452	359,313	–	–	**359,629**	359,419.53
7	371,193	**371,193**	**371,084.9**	370,999	370,815	–	–	**371,193**	370,874.10
8	351,994	**351,994**	**351,844.6**	351,836	351,001	–	–	**351,994**	351,800.20
9	349,337	**349,337**	**349,253.3**	348,732	348,309	–	–	**349,337**	349,140.97
10	351,415	**351,415**	351,125.6	351,408	351,415	–	–	**351,415**	351,144.80
Av.	360,460.9	360,460.9	360,339	360,316.6	360,040.2			360,460.9	360,279.80
Deviation%		0	0.03382	0.04003	0.11671			0	0.05024
bqp2500-1	1,515,944	**1,515,944**	1,514,804.6	1,514,971	1,515,011	-	–	1,515,807	1515518.30
2	1,471,392	1,471,357.8	1,469,721.0	1,468,694	1,468,850	–	–	1,471,106	1,470,316.07
3	1,414,192	1,414,183.1	1,412,943.0	1,410,721	1,413,083	–	–	**1,414,192**	1,413,492.13
4	1,507,701	**1,507,701**	**1,507,674.2**	1,506,242	1,506,943	–	–	**1,507,701**	1,507,324.60
5	1,491,816	**1,491,816**	**1,491,623.4**	1,491,796	1,491,465	–	–	1,491,770	1,491,473.60
6	1,469,162	**1,469,162**	1,467,918.2	1,467,700	1,468,427	–	–	1,468,977	1,468,179.57
7	1,479,040	**1,479,040**	1,477,101.7	1,476,059	1,478,654	–	–	1,478,844	1,478,204.77
8	1,484,199	**1,484,199**	1,483,226.9	1,484,199	1,482,953	–	–	1,484,159	1,483,613.47
9	1,482,413	**1,482,413**	1,481,622.9	1,482,306	1,481,834	–	–	1,482,302	1,482,000.93
10	1,483,355	1,483,336.9	1,481,899.2	1,482,354	1,482,166	–	–	1,483,122	1,482,155.33
Av.	1,479,921.4	1,479,915.28	1,478,853.51	1,478,504.2	1,478,938.6			1,479,798	1,479,227.877
Deviation%		0.00042	0.07216	0.09576	0.06641			0.00834	0.04686

Table 6
Comparison of the results obtained by the five metaheuristic methods and the DHNN–EDA for the 21 test problem instances with larger size and higher density.

Problem	Best known solution	ITS		MST1		MST2		MA-MK		DHNN–EDA		
		Best	Av.	Best	Av.	Best	Av.	Best	Av.	Best	Av.	Std.
p3000-1	3,931,583	0	0	0	667	0	0	3590	4784	0	1246.97	1324.34
p3000-2	5,193,073	0	97	0	117	0	97	342	1198	224	1671.87	863.01
p3000-3	5,111,533	0	344	357	897	0	287	0	3879	0	1806.97	1309.38
p3000-4	5,761,822	0	154	0	335	0	77	1097	2760	681	2316.40	1158.65
p3000-5	5,675,625	0	501	478	1154	0	382	478	2982	404	2462.87	1018.23
p4000-1	6,181,830	0	0	0	517	0	0	2390	3621	100	3204.30	1096.35
p4000-2	7,801,355	0	1285	1686	3597	0	804	6564	7870	2336	5066.27	1615.38
p4000-3	7,741,685	0	471	54	1465	0	1284	5760	7218	54	3135.60	1806.75
p4000-4	8,711,822	0	438	0	1246	0	667	2359	4995	1255	4660.43	1479.35
p4000-5	8,908,979	0	572	0	2611	0	717	9028	9567	923	6274.43	2173.55
p5000-1	8,559,355	375	646	2691	3893	0	256	4647	8173	1919	4996.77	2081.88
p5000-2	10,836,019	0	1068	0	3540	582	978	7519	10,790	1576	6381.83	2240.01
p5000-3	10,489,137	0	1266	3277	5644	0	1874	11,552	14,663	813	7136.93	3098.87
p5000-4	12,252,318	934	1952	3785	6714	1643	2570	16,399	18,188	1748	7643.63	3016.27
p5000-5	12,731,803	0	835	5150	7320	0	1233	6644	12,996	1655	5983.87	2457.40
p6000-1	11,384,976	0	57	3198	9213	0	34	9046	15,287	453	9231.47	4645.83
p6000-2	14,333,855	88	1709	10,001	11,626	0	1269	21,732	23,632	4329	9030.50	3401.94
p6000-3	16,132,915	2729	3064	11,658	14,958	0	2673	13,400	19,916	4464	10702.40	3660.03
p7000-1	14,478,676	340	1139	7118	9621	1607	2515	13,705	18,638	4529	10397.77	3539.01
p7000-2	18,249,948	1651	4301	8902	13,565	2330	3814	20,549	25,495	5750	14625.90	3809.74
p7000-3	20,446,407	0	3078	17,652	22,990	0	7868	14,684	29,584	1707	14002.63	8271.66
Av.	10,234,034.1	291.29	1094.14	3619.38	5794.76	293.43	1399.95	8165.95	11,725.52	1662.86	6284.75	2574.65
Deviation%	-	0.00285	0.01069	0.03537	0.05662	0.00287	0.01368	0.07979	0.11457	0.01624	0.06141	-

complexity of the perturbed operator in the ITS is $O(n^2)$. There are five parameters to be tuned in the ITS.

The MST1 and MST2 also have construction and improvement phase and all use the computation trick in solution improvement phases. In the construction phase, the MST1 uses a random restart and the MST2 uses a constructive procedure to a projected problem.

Table 6 displays the name of the instances, the best known-solution values, the results produced by the ITS, MST1, MST2, MA-MK and DHNN–EDA, respectively. The results of the competitors are directly adopted from the original papers of the competitors. For lager size instances, the absolute values of the solutions are very large, therefore, we report the relative values of the solutions like in Palubeckis (2006), that is, "Best" means the deviation from the best-known solution value of the best solution value found by the algorithms, and "Av." means the deviation from the best-known solution value of the average solution value found by the algorithms.

From Table 6, we can find that the DHNN–EDA outperforms the MA-MK. The average solution quality of the DHNN–EDA is even better than the best solution quality of the MA-MK. It suggests the MA-MK for problems with larger size and higher density is not as efficient as for problems with smaller size and lower density, and in contrast, the DHNN–EDA is seems to be more reliable and robust. Merz & Katayama (2004) claimed that larger instances with high density are difficult for the evolutionary algorithm to solve. The crossover in the MA-MK is designed by taking the properties of the fitness landscape into account for only the first group benchmark instances, and thus may be not suitable for the second group benchmark instances with larger size and higher density.

Comparing with the MST1, the DHNN–EDA obtains better solution quality in terms of best solution, and comparable average solution quality in the whole. But the DHNN–EDA is outperformed by the sophisticated designer algorithms, ITS and MST2.

4.6. 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 by not only CPU but also implementation language, the complier, RAM, operating system, even the programmer's skills. As done in many works (Cre'put & Koukam, 2009; Lan & DePuy, 2006), the actual computation times 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 (2007). Thus, the Mflop/s ratios called Dongarra's factors may be used to convert the CPU times of different machines. In this paper, we considered the same computer family for which performances are reported in

Table 7
Estimated performances of computers.

Computer	Performance estimated from Dongarra's paper (Mflop/s)	Estimated Dongarra's factor
UltraSPARC-Iii 440 MHz	202	0.1473
PC Pentium II 300 MHz	69	0.0503
R4000 CPU 100 MHz	15	0.0109
Silicon Graphics INDY R10,000sc 195 MHz	114	0.0832
Pentium III 800 MHz	165	0.1204
Pentium 4 2.8 GHz	1371	1

Table 8
Comparison of average computation time for 35 benchmark test problems.

Problem	MA-MK		MA-MF		TS-B		SA-B		EA	DHNN–EDA	
	C, Sun Ultra 5/10 (UltraSPARC-iii 440 MHz) Solaris 8	NT	C++, PC Pentium II 300 MHz Solaris 2.6	NT	R4000 CPU 100 MHz	NT	R4000 CPU 100 MHz	NT	Silicon Graphics INDY R10,000sc 195 MHz	NT	C, PC Pentium 4 2.8 GHz 512 MB
glov500	60	8.84	120	6.04	–	–	–	–	60 s	4.99	7.38
kb-g1000	360	53.03	–	–	–	–	–	–	–	–	34.67
bqp1000	360	53.03	600	30.18	4500	49.05	6800	74.12	–	–	34.10
bqp2500	360	53.03	1200	60.36	14 h	549.36	17 h	667.08	–	–	276.95

Dongarra's paper, and estimated the performance for the required processor frequency using interpolation. Table 7 presents the estimated performances interpolated at the required chip frequency, and the Dongarra's factors that will be used to compare the computation times.

Certainly, such factors lead to a crude computation time approximation. Lan & DePuy (2006) pointed out that the average error associated with using the Dongarra conversion could exceed 50% in some cases. Further, Johnson & McGeoch (2002) pointed 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 Operations Research, 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 (NT) in second to our PC (Pentium 4 2.80 GHz) in Tables 8 and 9.

From the rough estimation results shown in Table 8, we can find the DHNN–EDA is faster than the SA-B and TS-B, but worse than the MA-MF. The DHNN–EDA is faster than the MA-MK for 500 and 1000 variable instances, but slower for bqp2500.

From Table 9, we can find the DHNN–EDA spends more time than the ITS, MST1-2 and MA for the larger instances. In the ITS and MST1-2 for the UBQP, the complexity of the STS procedure decreases from $O(n^2)$ to $O(n)$ by using the computational trick mentioned above. Therefore, the ITS, MST1-2 algorithms based on the computational trick are all faster than the DHNN–EDA. In the ITS, MST1-2 algorithms, the stopping criterion is based on the CPU clock. Thus, given a fixed execution time for whole program, an open problem of these algorithms is how to get optimal trade-off between the number of TS restarts and the number of iterations of a TS run (Palubeckis, 2006). In the DHNN–EDA, HNN search procedure can quickly reach a stable state within a constant number of iterations, therefore, the DHNN–EDA can get good balance between the number of HNN search and solution quality more easily.

Given more time to the MA-MK, it is expected that the MA-MK can evolve more generations and produce some improvements in its performance. But Palubeckis in Palubeckis (2006) pointed out that increasing the time limit for the MA-MK several times does not help very much for the larger instances. The reason may be that the MA-MK is easily trapped in premature situation and thus can not improve the solution continuously for larger instances.

Note that Martı' (2003) also pointed out that one can not take the normalized results shown in Tables 8 and 9 too literally since those running times obtained on different machines. Small differences in running time should not be used to draw any conclusion regarding the relative effectiveness of the algorithms. Within the same order of magnitude should be regarded as indistinguishable (Martı', 2003).

From all the results mentioned above, we can conclude that the DHNN–EDA can obtain better or competitive results than metaheuristic algorithms within reasonable computation time.

Table 9
Comparison of average computation time for 21 benchmark test problems with larger size and higher density.

Problem	ITS, MST1, MST2, MA-MK		DHNN–EDA
	C, PC Pentium III 800 MHz	NT	C, PC Pentium 4 2.8 GHz 512 MB
p3000	900	108.36	558.32
p4000	1800	216.72	1001.58
p5000	3600	433.44	1572.40
p6000	5400	650.16	2377.82
p7000	9000	1083.60	3215.80

4.7. Discussion and remarks

In this section, firstly we 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 DHNN–EDA. Then, we explain our previous work of the DHNN based on the EDA and its application to the TPCNP in order to show the robustness of the proposed algorithm in this paper. Further, comparison of the DHNN–EDA with LS algorithms with EDA is also given. Finally, we discuss the DHNN–EDA in a multi-start algorithm framework and clarify the similarities and differences between the DHNN–EDA and other metaheuristic algorithms, which suggests or enlightens several ideas to further improve the performance of the DHNN–EDA.

4.7.1. Similarities and differences between chaotic neural models and DHNN–EDA

In order to deal with the local minima of the continuous HNN, Chen & Aihara (1995) proposed a TCNN by adding a decaying negative self-feedback to continuous HNN and thus introducing chaotic dynamics. Recently, Wang et al. (2004, 2008) proposed a NCNN by adding decaying stochastic noise into the TCNN. The chaotic dynamics and stochastic dynamics can prevent the network from getting stuck at local minima. But it is difficult to control and balance the chaotic dynamics, stochastic dynamics and gradient ascent dynamics for converging to a stable equilibrium point corresponding to an acceptably near-optimal solution. There are a lot of parameters to be tuned in these models to control those dynamics and the interaction of control parameters with the model performance is complex in practice.

In contrast to the TCNN and NCNN, the DHNN–EDA does not introduce the stochastic dynamics 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.

The DHNN–EDA has also deterministic dynamics like in the TCNN and is not guaranteed to converge to a global optimum. But from our simulation, the DHNN–EDA can obtain good solutions within reasonable computation time.

4.7.2. Explanation of our previous work of DHNN based on EDA

In our previous brief work (Wang, 2008), a similar DHNN approach based on EDA is proposed for the TPCNP. Simulation results show that the EDA can help the network escape from local minima, and thus the DHNN based on EDA algorithm outperforms the other NN algorithms and heuristics specifically designed for the TPCNP, which shows the robustness of the proposed algorithm in this paper.

However, the DHNN–EDA proposed in this paper has some difference from the previous work (Wang, 2008). In the previous work, the EDA mutates the best-so-far solution **gb** to generate a new starting point for HNN search like in the guided mutation (Zhang et al., 2004, 2005; Zhang et al., 2007; Zhang & Sun, 2006). In the work of this paper, the EDA mutates the current solution **v**. The current solution **v** appears to be rather good and not much worse than the best-so-far solution **gb.** The strategy to submit current solution **v** instead of best-so-far solution **gb** to the EDA mutation increase the diversification level in the search process, and the perturbed solution remains to be of sufficiently high quality and can serve as a good starting point for the next HNN search. Since this strategy increases the diversification in the search process, it can further improve the exploration performance of the DHNN based on EDA and thus find better solution finally, which is confirmed by our preliminary experiment results. Hence, the DHNN–EDA in this paper is an improved version of our previous algorithm. Further, more comprehensive comparison with other algorithms and analysis of the DHNN–EDA are provided in this paper.

4.7.3. Comparison of DHNN–EDA with LS algorithms with EDA

As mentioned in Introduction, some LS algorithms, including randomized first 1-opt LS, best improvement 1-opt LS and randomized k-opt LS, are proposed for the UBQP (Katayama, Yamashita, & Narihisa, 2007; Merz & Freisleben, 2002; Merz & Katayama, 2004). In order to confirm the effectiveness of the DHNN, we compare the DHNN–EDA with these LS algorithms combined with EDA mutation, respectively.

In the 1-opt LS algorithms, the single bit that delivers positive gain or the highest gain in objective function is flipped until the gain becomes negative in each iteration. After each flip, a vector of gains is maintained and updated. The basic idea of the k-opt LS is to find a solution by flipping a variable number k bits in the solution vector each iteration. In each step, a sequence of n solutions is generated by flipping the bit with the highest associated gain. After each flip, a vector of gains is also maintained and updated. Further, a candidate set is used to assure that each bit is flipped no more than once. The best solution in the sequence is accepted as the new solution for the next iteration. More specifically, in the randomized k-opt LS, the randomized first 1-opt LS is executed before the highest gain found by the best improvement 1-opt LS with a tabu fashion, which is to achieve a variable k-opt neighbor solution, is negative. Since the best improvement 1-opt with a tabu fashion can lead the randomized k-opt LS to temporally move to tentative solutions with negative gains (that is, permitting occasional bad moves), the randomized k-opt LS has ability of escaping from local minima, and is thus a more sophisticated LS method. The implementation of the randomized k-opt LS is also more sophisticated than that of the DHNN defined by Eqs. (4) and (5).

In the experiments, we used a restricted set of instances including kb-g1000-9, kb-g1000-10, bqp2500-2, bqp2500-3, bqp2500-10, p7000-1, p7000-2 and p7000-3. These instances are selected because, firstly, they are of different sizes and densities. Secondly, they are the hardest instances and thus can reveal the differences among the algorithms (Merz & Freisleben, 2002; Merz & Katayama, 2004; Palubeckis, 2006). In the pre-experimental testing, we find that the LS algorithms with EDA can obtain good performance by setting the same values of parameter β and λ as in the DHNN–EDA. Further, the same running time as in the DHNN–EDA is allotted for the implementation of the LS algorithms with EDA for fair comparison. Table 10 displays the name of the instances, the best-known solution value, the results produced by the randomized first 1-opt LS with EDA, the best improvement 1-opt LS with EDA, the randomized k-opt LS with EDA and the DHNN–EDA over 30 independent runs. For large size problems (p7000 instances), the relative values of the solutions are reported like in Table 6.

From Table 10, we can find that the DHNN–EDA is better than the best improvement 1-opt LS with EDA. The DHNN–EDA is also better than the randomized first 1-opt LS with EDA in the whole. For kb-g1000-9 and kb-g1000010 instances, the DHNN–EDA obtains slightly better average results than the randomized k-opt LS with EDA. However, for the other instances, the DHNN–EDA is outperformed by the randomized k-opt LS with EDA. We remark that it is reasonable because the randomized k-opt LS with the tabu fashion permits occasional bad moves to escape from local minima and thus shares some characteristics with complex metaheuristic. We also can find the randomized k-opt LS with EDA is better than

Table 10
Comparison of the results obtained by several LS methods combined with EDA and the DHNN–EDA.

Problem	Best known solution	Randomized first 1-opt LS with EDA			Best improvement 1-opt LS with EDA		
		Best	Av.	Std.	Best	Av.	Std.
kb-g1000-9	262,658	**262,658**	262,226.63	257.54	261,027	257,824.90	2316.64
kb-g1000-10	274,375	**274,375**	274,015.03	271.67	273,920	270,311.50	2123.40
bqp2500-2	1,471,392	1,470,788	1,470,227.53	289.24	1,470,624	1,468,758.40	1147.97
bqp2500-3	1,414,192	1,414,000	1,413,468.13	214.17	1,413,620	1,411,742.10	1237.15
bqp2500-10	1,483,355	1,482,794	1,482,108.50	282.47	1,482,586	1,480,616.20	1370.21
p7000-1	14,478,676	6278	8850.00	1881.33	25,398	53,984.00	16222.60
p7000-2	18,249,948	8347	12,177.00	2382.39	39,051	67,992.30	17,816.52
p7000-3	20,446,407	1909	9876.60	9001.38	50,081	98,622.30	23,669.15
		Randomized *k*-opt LS with EDA			DHNN–EDA		
		Best	Av.	Std.	Best	Av.	Std.
kb-g1000-9	262,658	**262,658**	262,058.73	311.36	**262,658**	262,391.57	172.84
kb-g1000-10	274,375	**274,375**	274,059.00	302.12	**274,375**	274,093.60	251.44
bqp2500-2	1,471,392	**1,471,392**	1,471,145.66	169.84	1,471,106	1,470,316.07	344.69
bqp2500-3	1,414,192	**1,414,192**	1,413,953.73	302.17	**1,414,192**	1,413,492.13	247.17
bqp2500-10	1,483,355	**1,483,355**	1,482,871.36	290.94	1,483,122	1,482,155.33	388.79
p7000-1	14,478,676	1346	4761.20	2379.67	4529	10,397.77	3539.01
p7000-2	18,249,948	2754	7080.00	2631.83	5750	14,625.90	3809.74
p7000-3	20,446,407	1256	13,776.00	6384.86	1707	14,002.63	8271.66

the MA-MK for p7000 instances. As pointed out earlier, the reason may be due to that the crossover in the MA-MK is designed by taking the properties of the fitness landscape into account for only the first group benchmark instances, and may be not suitable for the second group benchmark instances with larger size and higher density. Thus, the MA-MK is easily trapped in premature situation. In contrast, EDA can effectively guide the randomized *k*-opt LS to search better results.

Katayama et al. (2007) pointed out that the performance of LS highly depends on neighborhood definitions and replacement ways and thus problem-specific neighborhood operator (flip, swap, or add and drop operator) should be designed carefully for specific problem. For HNNs, only the problem-specific objective function is embedded in the energy function that the network evolution tries to minimize. For a specific problem, simple effective LS can also be incorporated into standard HNNs. For example, an efficient multivalued HNN was proposed for the traveling salesman problem (TSP) (Mérida-Casermeiro, Galán-Marín, & Muñoz-Peréz, 2001). The dynamics or updating rule for the multivalued HNN guaranteeing the energy decrease are derived from the efficient *k*-opt (2-opt and 3-opt) moves. Simulation results show that the multivalued HNN is superior to the best NN for the TSP at that time. Thus, it is concluded in Mérida-Casermeiro et al. (2001) that for NNs the hybrids with classical optimization techniques are one of the best ways to go if one wants to handle larger instances and obtain reasonably good results. More recently, a constructive-optimizer neural network (CONN) was proposed for the TSP (Saadatmand-Tarzjan, Khademi, Akbarzadeh-T, & Moghaddam, 2007). The basic optimizer NN (BONN) in the CONN is algorithmically similar to the family of 2.5-opt LS for the TSP. The BONN uses a feedback configuration similar to HNNs and is presented in the context of NNs. The energy function decreases monotonically during BONN training. Simulation results show that the CONN provides the best compromise between computation time and accuracy among currently reported NNs for the TSP. Hence, the randomized *k*-opt LS can also be incorporated into the DHNN–EDA, which is just our future research work.

4.7.4. Similarities and differences between DHNN–EDA and metaheuristic algorithms

The ITS, MA-MK, MST1-2, and the DHNN–EDA all can be seen as multi-start 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 these algorithms (Martı', 2003). In the solution construction phase, a simple strategy, for example, random generation of start solution can be used. Similarly, a simple solution improvement method can be applied, for example, simple LS. Further, the solution improvement phase can be performed with a complex metaheuristic, for example, SA and TS.

In order to better understand the inner instrument of these algorithms concluding the ITS, MA-MK, MST1-2, and the DHNN–EDA and the commonalities and differences among them, we first introduce a three-dimensional categorizing framework. Solution construction and solution improvement are considered as two dimensions in the framework. Under solution construction strategies, algorithms are categorized into constructive and perturbation based methods. Under solution improvement strategies, algorithms are categorized into simple LS and complex metaheuristic. We consider population or single point based strategy as a third dimension. This categorizing framework in this paper serves two roles. First, it reveals relationships among different algorithms. Second, it also reveals what are missing in the current combinations of different strategies and thus indicates potential new or improved algorithms with suitable combinations of different strategies in different dimension.

The ITS uses the computational trick designed for the UBQP in both solution perturbation (construction) and solution improvement phases. It is a single point, perturbation (perturbation GSP) and complex metaheuristic (metaheuristic STS) based algorithm. The MST1 and MST2 also use the computation trick in solution improvement phases. Further, the MST2 uses a constructive procedure to a projected problem in the solution construction phase. Therefore, the MST1 and MST2 are all single point, constructive method and complex metaheuristic-based algorithms. The MA-MK uses an innovative variation (crossover operator) to construct or generate a population of offspring solutions, a diversification/restart strategy (mutation operator) to mutate or perturb each offspring, and the randomized *k*-opt LS to improve the offspring solutions at each generation. Obviously, the MA-MK is a population-based algorithm. The crossover operator can be seen as a constructive method and the mutation operator is obviously a

perturbation based method. Though randomized *k*-opt LS is not a metaheuristic methods, it is a more sophisticated LS. The innovative variation and randomized *k*-opt LS are all tailed for the UBQP. Thus, the MA-MK can be seen as a hybrid method. If the population size of the MA-MK is set to 1, the MA-MK has no crossover operator and the mutation operator and randomized *k*-opt LS are fully used to generate new offspring. Thus it becomes to a single point, perturbation and LS based algorithm.

All the metaheuristic algorithms mentioned above use more sophisticated procedures specifically tailored for the UBQP. Misevcius, Smolinskas, & Tomkevičius (2005) pointed out that such "tailored" algorithms, like the ITS, MST1-2 and MA-MK, 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 multi-started based algorithms are problem-dependent and the ideas and strategies implemented are difficult to apply directly to different problems.

In the DHNN–EDA, the solution construction phase is based on EDA, and the solution improvement phase is based on HNN, a kind of simple and general LS. It is a single point, perturbation and simple LS based algorithm. Thus, the DHNN–EDA is based on a simple and efficient framework that can be used directly to design solving methods for other combinatorial optimization problems, for example, the TPCNP in our previous work (Wang, 2008). In summary, the DHNN–EDA is based on a much simpler and more general-purpose approach. Note that it may be argued that the HNN is based on gradient descent and thus also uses the information from the problem structure implicitly. However, the network derives the gradient information of the problem in a general way described by the motion Eq. (4), which is not necessary to be designed specifically for different problems.

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 view as a machine which consist of large number of simple interconnected processing units and therefore, can implement complex computation 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 analogue or digital integrated circuit or optical computers (Dominguez & Munoz, 2008; Huang & Wang, 2003; Resende & Werneck, 2004; Smitha et al., 2003). Considering the speed advantage, it is foreseeable that NNs will soon become a rapid solution technique for large and complex combinatorial optimization problems (Smitha et al., 2003).

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

4.7.5. Future research lines for further improvement of DHNN–EDA

From the experiments, it is clear that the DHNN–EDA is not the best algorithm for the UBQP. According to the three-dimensional categorizing framework discussed above, we can borrow some ideas from the competitors or other metaheuristic algorithms and invest more intellectual effort by the following ways to further improve the performance of the DHNN–EDA.

Firstly, 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 (Battiti & Brunato, 2007). Another way of adapting the perturbation is to change deterministically its strength during the search, just as done in variable neighborhood search (VNS) (Hansen & Mladenovic, 2003).

Secondly, population-based DHNN–EDA extensions. In the population-based DHNN–EDA, EDA guides the search and a population of starting points are kept and used to run the HNN search. Thus, the population-based DHNN–EDA extensions share similarities to the MA. In general, population-based algorithms are more complex to use than single point based algorithms, and require mechanisms to manage a population of solutions. More importantly in the MA, it is necessary to design effective "crossover" operator for the combination of solutions in the population and effective "mutation" operator or other mechanisms to keep the diversity of the population, which, more often, is a challenge and key task. For example, a crossover operator, innovative variation, and a diversification/restart strategy in the MA-MK is specifically designed or tailored to obtain that performance of the algorithm for the first group benchmark instances. Further, one question arises is whether using a population of solutions is really helpful. At least, it is pointed out that the single point based search algorithm leads to good results for some problems such as TSP with high cost-distance correlations, and thus the advantage of population-based method is small or nil (Lourenço, Martin, & Stützle, 2003). For some other problems, the population-based algorithms can achieve diversification in search process and thus are desirable if their complexity is not overwhelming. These problems can be carefully studied in detail in future research.

Finally, adaptive memory or mechanism to escape from local minima in the HNN descent. In the DHNN–EDA, the solution improvement phase is based on a simple HNN search. Inspired by the ITS and the randomized *k*-opt LS, the tabu mechanism (tabu list) explicitly using the history of the search can be incorporated into the HNN to further improve the performance of the DHNN–EDA. Specifically, a tabu search rule, instead of the original updating rule described by Eqs. (4) and (5) in the DHNN, is used to govern the state transition of neurons to search for the global minimum of the energy function, and thus can help the network escape from local minima. It is a promising approach to introduce the tabu mechanism into the DHNN and propose a tabu NN with EDA algorithm. Thus, the short term memory of the tabu mechanism in the tabu NN cooperates with the long term memory mechanism in the EDA to improve the performance of the DHNN–EDA, which is just our ongoing work.

Note that ideas mentioned above can further improve the performance of the DHNN–EDA, but the modified DHNN–EDA with those ideas may become more and more sophisticated and thus run the risk of losing simplicity.

5. Conclusions

In this paper, we have presented a novel DHNN–EDA as a competitive approach for the UBQP. In the DHNN–EDA, the idea of EDA is incorporated into the DHNN in order to overcome the local minima problem of the DHNN. The perturbation based on the EDA in the DHNN–EDA can generate a new starting point for the DHNN for further search. Therefore, the DHNN–EDA can escape from local minima and further search better results. Simulation results on the UBQP show that the DHNN–EDA is better than the other improved HNN algorithms such as multi-start DHNN and DHNN with random flips, and is better or competitive with metaheuristic algorithms such as SA, TS, SS and MA. Further, we discuss the DHNN–EDA in the multi-start algorithm framework and clarify the similarities and differences between the DHNN–EDA and other metaheuristic algorithms. We also mention the TCNN and NCNN, and discuss the similarities and differences between the chaotic neural models and the DHNN–EDA. Based on these discussions,

we highlight the characteristics of the DHNN–EDA and propose some ways to further improve the performance of the DHNN–EDA.

In the future, several research lines can be followed. Firstly, the simulation results for the UBQP, together with the results for the TPCNP (Wang, 2008), show that the DHNN–EDA is an efficient and robust combinatorial optimization framework and therefore, can be applied to other combinatorial optimization problems. Secondly, the perturbation based on EDA in the DHNN–EDA can be incorporated into other DHNN models, for example, competitive HNN (Galán-Marín, Mérida-Casermeiro, & Muñoz-Pérez, 2003; Galán-Marín & Muñoz-Pérez, 2001), hysteretic HNN (Xia, Tang, Li, & Wang, 2005), and quantized HNN (Nourelfath & Nahas, 2003), for solving different kinds of combinatorial optimization problems. Finally and most importantly, we will borrow some ideas from the competitors or other metaheuristic algorithms following the ways mentioned in the discussion section to further improve the performance of the DHNN–EDA.

Acknowledgments

This work was supported in part by the National Natural Science Foundation of China (60805026, 60905038, 60873162), the Specialized Research Fund for the Doctoral Program of Higher Education (20070558052), and the Scientific Research Foundation for the Returned Overseas Chinese Scholars, State Education Ministry (2007-1108).

References

Alkhamis, T. M., Hasab, M., & Ahmed, M. A. (1998). Simulated annealing for the unconstrained quadratic pseudo-boolean function. *European Journal of Operational Research, 108*, 641–652.

Amini, M. M., Alidaee, B., & Kochemberger, G. A. (1999). A scatter search approach to unconstrained quadratic binary programs. In D. Corne, M. Dorigo, & F. Glover (Eds.), *New ideas in optimisation* (pp. 317–329). London: McGraw-Hill.

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

Battiti, R., & Brunato, M. (2007). Reactive search: Machine learning for memory-based heuristics. In Teófilo Gonzalez (Ed.), *Handbook of approximation algorithms and metaheuristics*. *Computer and Information Science Series*. Chapman & Hall/CRC.

Beasley, J. E. (1998). *Heuristic algorithms for the unconstrained binary quadratic programming problem*. Technical report, Management School, Imperial College, London, UK.

Beasley, J. E. (2008). *OR-library: Unconstrained binary quadratic programming (Beasley, 1990 and Badics, 1996)*. <http://mscmga.ms.ic.ac.uk/jeb/orlib/bqpinfo.html>.

Blas, A. D., Jagota, A., & Hughey, R. (2005). Optimizing neural networks on SIMD parallel computers. *Parallel Computing, 31*(1), 97–115.

Boros, E., Hammer, P. L., & Tavares, G. (2007). Local search heuristics for quadratic unconstrained binary optimization (QUBO). *Journal of Heuristics, 13*, 99–132.

Chen, L., & Aihara, K. (1995). Chaotic simulated annealing by a neural network model with transient chaos. *Neural Networks, 8*(6), 915–930.

Cre'put, J.-C., & Koukam, A. (2009). A memetic neural network for the Euclidean traveling salesman problem. *Neurocomputing, 72*(4-6), 1250–1264.

Dominguez, E., & Munoz, J. (2008). A neural model for the *p*-media problem. *Computer & Operation Research, 35*, 404–416.

Dongarra, J. J. (2007). *Performance of various computers using standard linear equations software*. Technical Report CS-89-85, Department of Computer Science, University of Tennessee, USA, <http://www.netlib.org/benchmark/performance.ps>.

Funabiki, N., & Kitamichi, J. (1999). A gradual neural network algorithm for broadcast scheduling problems in packet radio networks. *IEICE Transactions on Fundamentals, E82-A*, 815–824.

Funabiki, N., & Nishikawa, S. (1997a). A binary Hopfield neural-network approach for satellite broadcast scheduling problems. *IEEE Transactions on Neural Networks, 8*, 441–445.

Funabiki, N., & Nishikawa, S. (1997b). A gradual neural-network approach for frequency assignment in satellite communication systems. *IEEE Transactions on Neural Networks, 8*, 1359–1370.

Funabiki, N., Takefuji, Y., & Lee, K. C. (1993). Comparison of six neural network models on a traffic control problem in a multistage interconnection network. *IEEE Transactions on Computers, 42*, 497–501.

Galán-Marín, G., Mérida-Casermeiro, E., & Muñoz-Pérez, J. (2003). Modelling competitive Hopfield networks for the maximum clique problem. *Computer & Operations Research, 30*(4), 603–624.

Galán-Marín, G., & Muñoz-Pérez, J. (2001). Design and analysis of maximum Hopfield networks. *IEEE Transactions on Neural Networks, 12*(2), 329–339.

Garey, M. R., & Johnson, D. S. (1979). *Computers and intractability: A guide to the theory of NP-completeness*. New York: Freeman.

Glover, F., Alidaee, B., Rego, C., & Kochenberger, G. (2002). One-pass heuristics for large-scale unconstrained binary quadratic problems. *European Journal of Operational Research, 137*, 272–287.

Glover, F., Kochenberger, G., & Alidaee, B. (1998). Adaptative memory tabu search for binary quadratic programs. *Management Science, 44*(3), 336–345.

Hansen, P., & Mladenovic, N. (2003). Variable neighborhood search. In Glover & Kochenberger (Eds.), *Handbook of metaheuristics* (pp. 145–184). Berlin: Springer.

Harik, G. R., Lobo, F. G., & Goldberg, D. E. (1999). The compact genetic algorithm. *IEEE Transactions on Evolutionary Computation, 3*(4), 287–297.

Hasab, M., Alkhamis, T., & Ali, J. (2000). A comparison between simulated annealing, genetic algorithm and tabu search methods for the unconstrained quadratic pseudo-boolean function. *Computers & Industrial Engineering, 38*, 323–340.

He, H. & Sykora, O. (2006). A Hopfield neural network model for the outerplanar drawing problem. In *Proceedings of the international multi conference of engineers and computer scientists 2006 (IMECS '06), Hong Kong, China* (pp. 42–47).

Helmberg, C., & Rendl, F. (1998). Solving quadratic (0,1)-problems by semidefinite programs and cutting planes. *Mathematical Programming, 82*, 291–315.

He, H., Sykora, O., & Makinen, E. (2006). An improved neural network model for the two-page crossing number problem. *IEEE Transactions on Neural Networks, 17*(6), 1642–1646.

Hopfield, J. J. (1982). Neural networks and physical systems with emergent collective computational abilities. *Proceedings of National Academy of Sciences, 79*, 2554–2558.

Hopfield, J. J. (1984). Neurons with graded response have collective computational properties like those of two-state neurons. *Proceedings of National Academy of Sciences, 81*, 3088–3092.

Hopfield, J. J., & Tank, D. W. (1985). Neural computation of decisions in optimization problems. *Biological Cybernetics, 52*, 141–152.

Huang, D. L., & Wang, J. (2003). Digital hardware realization of a recurrent neural network for solving the assignment problem. *Neurocomputing, 51*, 447–461.

Johnson, D. S., & McGeoch, L. A. (2002). Experimental analysis of heuristics for the STSP. In G. Gutin & A. Punnen (Eds.), *Traveling salesman problem and its variations* (pp. 369–443). Dordrecht: Kluwer Academic Publishers.

Katayama, K., Yamashita, H. & Narihisa, H. (2007). Variable depth search and iterated local search for the node placement problem in multihop WDM lightwave networks. In *Proceedings of 2007 IEEE congress on evolutionary computation* (pp. 3508–3515).

Katayama, K., & Narihisa, H. (2001). Performance of simulated annealing-based heuristic for the unconstrained binary quadratic programming problem. *European Journal of Operational Research, 134*, 103–119.

Kochenberger, G., Glover, F., Alidaee, B., & Rego, C. (2004). A unified modeling and solution framework for combinatorial optimization problems. *OR Spectrum, 26*, 1–14.

Lan, G., & DePuy, G. W. (2006). On the effectiveness of incorporating randomness and memory into a multi-start metaheuristic with application to the set covering problem. *Computers & Industrial Engineering, 51*, 362–374.

Larranaga, P., & Lozano, J. (2001). *Estimation of distribution algorithms: A new tool for evolutionary computation. Genetic algorithms and evolutionary computation* (Vol. 2). Springer.

Lodi, A., Allemand, K., & Liebling, T. M. (1999). An evolutionary heuristic for quadratic 0-1 programming. *European Journal of Operational Research, 119*, 662–670.

Lourenço, H. R., Martin, O. C., & Stützle, T. (2003). Iterated local search. In Glover & Kochenberger (Eds.), *Handbook of metaheuristics* (pp. 321–354). Berlin: Springer.

Marti', R. (2003). Multi-start methods. In Glover & Kochenberger (Eds.), *Handbook of metaheuristics* (pp. 355–368). Berlin: Springer.

Mérida-Casermeiro, E., Galán-Marín, G., & Muñoz-Peréz, J. (2001). An efficient multivalued Hopfield network for the traveling salesman problem. *Neural Processing Letters, 14*(3), 203–214.

Merz, P., & Freisleben, B. (1999). Genetic algorithms for binary quadratic programming. In *Proceedings of the 1999 international genetic and evolutionary computation conference (GECCO'99)* (pp. 417–424).

Merz, P., & Freisleben, B. (2002). Greedy and local search heuristics for unconstrained binary quadratic programming. *Journal of Heuristics, 8*, 197–213.

Merz, P., & Katayama, K. (2004). Memetic algorithms for the unconstrained binary quadratic programming problem. *BioSystems, 78*, 99–118.

Misevicius, A., Smolinskas, J., & Tomkevičius, A. (2005). Using iterated tabu search for the traveling salesman problem: new results. *Information Technology and Control, 34*(4), 327–337.

Muhlenbein, H. & Paaß, G. (1996). From recombination of genes to the estimation of distributions. In H.-M. Voigt, W. Ebeling, I. Rechenberg, & H.-P. Schwefel (Eds.), Proceedings of the 4th conference on parallel problem solving from nature-PPSN IV. *Lecture notes in computer science* (Vol. 1411, pp. 178–187). Berlin: Springer.

Nourelfath, M., & Nahas, N. (2003). Quantized Hopfield networks for reliability optimization. *Reliability Engineering & System Safety, 81*(2), 191–196.

Palubeckis, G. (1995). A heuristic-based branch and bound algorithm for unconstrained quadratic zero-one programming. *Computing, 54*, 283–301.

Palubeckis, G. (2004). Multistart tabu search strategies for the unconstrained binary quadratic optimization problem. *Annals of Operations Research, 131*, 259–282.

Palubeckis, G. (2006). Iterated tabu search strategies for the unconstrained binary quadratic optimization problem. *Informatica, 17*(2), 279–296.

Pelikan, M., Goldberg, D. E., & Lobo, F. (2002). A survey of optimization by building and using probabilistic models. *Computational Optimization and Applications, 21*(1), 5–20.

Resende, M. G. C., & Werneck, R. F. (2004). A hybrid heuristic for the *p*-median problem. *Journal of Heuristics, 10*, 59–88.

Saadatmand-Tarzjan, M., Khademi, M., Akbarzadeh-T, M.-R., & Moghaddam, H. A. (2007). A novel constructive-optimizer neural network for the traveling salesman problem. *IEEE Transactions on System, Man, and Cybernetics – Part B: Cybernetics, 37*(4), 754–770.

Smitha, K. A., Abramsonb, D., & Duke, D. (2003). Hopfield neural networks for timetabling: Formulations, methods, and comparative results. *Computers & Industrial Engineering, 44*, 283–305.

Wang, J. (2008). Hopfield neural network based on estimation of distribution for two-page crossing number problem. *IEEE Transactions on on Circuits and Systems II: Express Briefs, 55*(8), 797–801.

Wang, L., Li, S., Tian, F., & Fu, X. (2004). A noisy chaotic neural network for solving combinatorial optimization problems: Stochastic chaotic simulated annealing. *IEEE Transactions on System, Man, and Cybernetics, Part B – Cybernetics, 34*(5), 2119–2125.

Wang, L., Liu, W., & Shi, H. (2008). Noisy chaotic neural networks with variable thresholds for the frequency assignment problem in satellite communications. *IEEE Transactions on System, Man, Cybernetics, Part C – Reviews and Applications, 38*(2), 209–217.

Xia, G., Tang, Z., Li, Y., & Wang, J. (2005). A binary Hopfield neural network with hysteresis for large crossbar packet-switches. *Neurocomputing, 67*, 417–425.

Zhang, Q., & Sun, J. (2006). Iterated local search with guided mutation. *Proceedings of IEEE Congress on Evolutionary Computation*, 924–929.

Zhang, Q., Sun, J., & Tsang, E. (2004). Combinations of estimation of distribution algorithms and other techniques. *International Journal of Automation & Computing, 4*(3), 273–280.

Zhang, Q., Sun, J., & Tsang, E. (2005). Evolutionary algorithm with guided mutation for the maximum clique problem. *IEEE Transactions on Evolutionary Computation, 9*(2), 192–200.

Zhang, Q., Sun, J., Xiao, G., & Tsang, E. (2007). Evolutionary algorithms refining a heuristic: A hybrid method for shared-path protections in WDM networks under SRLG constraints. *IEEE Transactions on Systems, Man and Cybernetics, Part B, 37*(1), 51–61.