Neural Process Lett (2008) 27:1–15
DOI 10.1007/s11063-007-9055-8

Hopfield Network as Static Optimizer: Learning the Weights and Eliminating the Guesswork

Gursel Serpen

Published online: 3 November 2007

Abstract This article presents a simulation study for validation of an adaptation methodology for learning weights of a Hopfield neural network configured as a static optimizer. The quadratic Liapunov function associated with the Hopfield network dynamics is leveraged to map the set of constraints associated with a static optimization problem. This approach leads to a set of constraint-specific penalty or weighting coefficients whose values need to be defined. The methodology leverages a learning-based approach to define values of constraint weighting coefficients through adaptation. These values are in turn used to compute values of network weights, effectively eliminating the guesswork in defining weight values for a given static optimization problem, which has been a long-standing challenge in artificial neural networks. The simulation study is performed using the Traveling Salesman problem from the domain of combinatorial optimization. Simulation results indicate that the adaptation procedure is able to guide the Hopfield network towards solutions of the problem starting with random values for weights and constraint weighting coefficients. At the conclusion of the adaptation phase, the Hopfield network acquires weight values which readily position the network to search for local minimum solutions. The demonstrated successful application of the adaptation procedure eliminates the need to guess or predetermine the values for weights of the Hopfield network.

Keywords Hopfield neural network · Static optimization · Combinatorial · Weights · Adaptation · Learning · Training · Traveling salesman problem · Computational complexity · Gradient descent · Liapunov function

G. Serpen (✉)
Electrical Engineering and Computer Science Department, The University of Toledo, Toledo, OH 43606, USA
e-mail: gserpen@eng.utoledo.edu

1 Introduction

Static optimization and more specifically the combinatorial optimization has been subject to rigorous empirical studies in numerous domains including hard and soft sciences, engineering, economics, business, and finance to list a few during the past decades particularly after the widespread emergence of mainframe computing in early 1970s. Most notable solutions leveraged direct search techniques in graph theory, heuristic search in artificial intelligence, and dynamic programming in operations research among others. Reasonably large-scale problem instances were successfully addressed for near-optimal solutions by these algorithms even though in some cases computational complexity, i.e. both spatial and temporal, turned out to be prohibitive for a real time context. Artificial neural network based optimizers on the other hand might be positioned to address this very same challenge; scalability with respect to real time computation requirements as the problem size increases to real life dimensions. This premise is based on the fact that neural network optimizers do offer the prospect of hardware realization upon appropriate technological advances and accordingly are poised to fully leverage the massive parallelism inherent to neural optimization algorithms. If in fact adequate technological advances can be realized, then the neural optimizers can compute solutions of hard optimization problems practically in constant time regardless of the problem size. This premise is inherently lacking in non-neural approaches: no other non-neural algorithm appears to offer an ability to leverage such a massive degree of parallelism so that it can be promising for a real time context.

The artificial neural network literature offers an array of neural optimization algorithms some of which are well positioned for real-time solutions of static optimization problems [1,2]. The Hopfield network [3] and its stochastic or chaotic derivatives are prominent static optimizers for addressing unconstrained, linear, quadratic programming, linear complementarity, discrete and combinatorial optimization problems. The Hopfield network, the most basic recurrent architecture for optimization, is a fundamental building block in this venue, is subject to substantial current interest [4,5], and is considered as an important continuous-time computation paradigm [6]. It employs a gradient descent search mechanism and its computational promise is to find a locally optimum solution. On the other hand its stochastic or chaotic derivatives, Boltzman machine with simulated annealing, mean field annealing network, and the chaotic Hopfield network [7], can compute near optimal solutions.

Recent theoretical developments exposed the computational power of the Hopfield network and the findings are very promising: a Hopfield network subject to certain mild conditions appears to be equivalent to a Turing machine as well as being able to address the NP-hard "minimum energy" problem in polynomial time [8–10]. This development establishing the continuous Hopfield network as a universal computing machine only adds to the excitement towards fully leveraging its computational promise. This might also explain the widespread ongoing interest among theoretical and applied researchers on the Hopfield network theory and its applications as indicated by the recent literature although not so many real-life applications of the Hopfield network appear to have been reported to date. A quick search on only two of the many citation databases, PapersFirst® and ArticleFirst®, suggested that there were over 800 papers, where the search query employed the two keywords "Hopfield" and "optimization" for the article title field since 2000. A good many of these articles were applications of the Hopfield network to static optimization problems in a surprising array of fields encompassing social sciences, economics, finance, business, geography, image and vision, forestry sciences, engineering, sciences, geography, as well as traditional artificial intelligence and operations research among those to name.

Hopfield network offers a true "real-time" optimization algorithm for computation of a local optimum solution of a static optimization problem assuming a hardware-centric implementation that takes advantage of the high-degree of inherent parallelism. The promise is a quick and local optimum solution, and scalability of the computation time with the increase in the size of the problem [1]. Although theoretically very promising, in practice the Hopfield neural network paradigm has been suffering from a number of important shortcomings. Hopfield network in its basic form fails to compute at times feasible or valid solutions for particularly large-scale static optimization problems. This shortcoming appears to be tied to the presence of stable equilibrium (or fixed) points in the state space of network dynamics as attractors which are not solutions of a particular optimization problem (i.e. infeasible solution points). Weights of the network dynamics play an important role in determining the stable equilibrium points and are typically defined through heuristics or empirical means in the basic form of the Hopfield network. The issue was addressed with partial success through stability-theoretic procedures to determine bounds on adaptable parameters, mainly constraint weighting coefficients, modifications to the network dynamics and the error equation among other countless fixes [11–14]. It was demonstrated that the techniques that precompute the values of weights fail to guide the Hopfield network dynamics towards feasible solutions for those problems for which only a small fraction of the stable equilibrium points are solutions [15]. A second and perhaps equally no less important weakness associated with Hopfield network algorithm was that the computational power was constrained to finding locally optimal solutions on the average by the virtue of employing a gradient descent search mechanism. This shortcoming has been addressed through stochastic or chaotic enhancements at the expense of relatively large computational cost or partial serialization [7].

Recently, a theoretical framework for an adaptive Hopfield network that attains its weight vector through training for a given static optimization problem was proposed [16]. However, the training has to be performed in an extremely high dimensional weight space, i.e. $O\left(N^4\right)$ dimensions for an N-vertex graph search problem, i.e. a 500-vertex graph would require 62,500,000,000 weights or, more precisely, half of this number if symmetry of the weight matrix is considered. It is well-known in machine learning that too many freely-adjustable parameters lead to a model that is ordinarily too complex [17]. Regularization techniques are often employed to reduce the number of freely adjustable parameters to facilitate a desirable learning profile [18–20]. Also the issue of ill-conditioning is of concern for the case of gradient-descent based algorithms since the direction for most efficient descent is much more difficult to determine, as a consequence of the high dimensionality of the surface [21].

Noting one peculiar aspect of Hopfield networks, this paper proposes that one is not forced to perform the adaptation in the original weight space. Instead, the training or learning can be performed in the comparably very low dimensional constraint weighting coefficient space, since it is possible to compute all the entries in a weight matrix once the constraint weighting coefficient values are known for a Hopfield network [22]. The study reported herein aims to empirically test and validate this conjecture to establish feasibility of and to finalize the implementation aspects of such an adaptation algorithm.

This paper presents a simulation study on learning-based adaptation of the Hopfield network as a static combinatorial optimizer in the constraint weighting coefficient space: the goal is determining the values of weights that will guide the network towards a local optimum solution for a given optimization problem. Initially, underlying principles of the theoretical and mathematical framework are discussed. Specifically, the classical Hopfield network, the computational procedure that determines values of weights given constraint weighting coefficients, the adaptation scheme which entails the gradient descent in the constraint weighting coefficient space, and the configuration of Hopfield network dynamics for static optimization

are discussed. All this is followed by the simulation study based on a combinatorial optimization problem (the traveling salesman), analysis of the findings, and the conclusions.

2 Hopfield Network and Adaptation

2.1 Hopfield Network

A Hopfield network is a nonlinear dynamical system [3], whereby the definition of the continuous Hopfield network is as follows. Let z_i represent a node output and $z_i \in [0, 1]$ with $i = 1, 2, \ldots, K$, where K is the number of network nodes. Then,

$$E = -\frac{1}{2}\sum_{i=1}^{K}\sum_{j=1}^{K} w_{ij} z_i z_j + \frac{1}{\lambda}\sum_{i=1}^{K}\int_{0}^{z_i} f^{-1}(z)dz - \sum_{i=1}^{K} b_i z_i \tag{1a}$$

is a Liapunov function for the system of equations defined by

$$\frac{du_i(t)}{dt} = -u_i(t) + \sum_{j=1}^{K} w_{ij} z_j(t) + b_i \quad \text{and} \quad z_i = f(u_i), \tag{1b}$$

where w_{ij} is the weight between nodes z_i and z_j subject to $w_{ij} = w_{ji}$, b_i is the external input for node z_i, and $f(\cdot)$ is a nonlinearity, typically the sigmoid function with steepness defined by a parameter λ. Note that the second term in the Liapunov function vanishes for very large positive values of the parameter λ for cases where the activation function is a sigmoid.

The set of formulas given in Eq. 1b seek out through gradient descent a local minimum of the quantity represented by Eq. 1a as dictated by the initial values of the neurons, neuron update scheme (i.e. asynchronous versus synchronous) and order, and the basins of attractions of stable equilibrium points in the state space of the network dynamics. It has been shown that, for a large class of static optimization problems including those that are graph-theoretic, many local minima in the Liapunov space may not be feasible solutions while those local minima associated with feasible solutions are far from the optimal solution [14,15].

Learning through experience, i.e. prior mistakes or attempts, is the only feasible option for Hopfield network as a static optimizer since no so-called training exemplars exist. Accordingly, an adaptation scheme needs to look at prior unsuccessful attempts to extract and integrate the associated experience into the network's search algorithm. In other words, an adaptation mechanism is needed that extracts the required information from previous unsuccessful search attempts in order to incorporate it into the neural network dynamics so that it is less likely for the Hopfield network to repeat the same failed search process or similar ones. The adaptation methodology as detailed in the next sections follows this overall line of reasoning to guide the Hopfield network towards feasible local minimum solutions of static optimization problems. It also strives to scale with the size of the problem in the sense that the applicability of adaptation procedure does not break down for larger problem instances.

2.2 Adaptation Scheme for Hopfield Network

A typical static optimization problem can be mapped to a single layer and often two-dimensional neuron topology of the Hopfield network through empirical development of an error function [3]. Each constraint of a static optimization problem can be mapped to the network topology using an error term, formulated as either a quadratic or linear sum as follows:

$$E_\varphi(\mathbf{z}) = g_\alpha \sum_{i=1}^{K}\sum_{j=1}^{K} d_{ij}^{\alpha}\delta_{ij}^{\alpha} z_i z_j \text{ or } E_\varphi(\mathbf{z}) = g_\beta \sum_{i=1}^{K} \delta_i^{\beta} z_i \quad \text{for } \varphi = 1, 2, \ldots, \left|S_\varphi\right|, \quad (2a)$$

where

a. $i \neq j$;
b. the set of constraints is given by $S_\varphi = \left\{C_1, C_2, \ldots, C_\varphi\right\}$ with S_α and S_β representing sets of quadratic and linearly formulated constraints, respectively, $S_\varphi = S_\alpha \cup S_\beta$, $S_\alpha \cap S_\beta = \phi$ and $\left|S_\varphi\right| = |S_\alpha| + \left|S_\beta\right|$;
c. $g_\varphi \in R^+$ if the hypothesis nodes z_i and z_j each represent for a constraint C_α in S_α are mutually supporting and $g_\varphi \in R^-$ if the same hypotheses are mutually conflicting; the term δ_{ij}^{α} is equal to 1 if the two hypotheses represented by nodes z_i and z_j are related under the constraint C_α and is equal to 0 otherwise;
d. similarly, the term δ_i^{β} is equal to 1 or 0 to facilitate mapping of the constraint C_β in S_β to the network topology;
e. and the d_{ij}^{α} term is equal to 1 for all i and j under a hard constraint (which cannot be violated by definition) and is a predefined cost for a soft constraint (which can be violated in degrees), which is typically associated with a cost term in optimization problems.

In the case of a typical static optimization problem that has multiple constraints, the error function can be defined as an algebraic sum of quadratic and linear error terms:

$$\begin{aligned} E(\mathbf{z}, t_k) &= E_1(\mathbf{z}, t_k) + E_2(\mathbf{z}, t_k) + \cdots + E_{|S_\varphi|}(\mathbf{z}, t_k) \\ &= g_1(t_k)E_1'(\mathbf{z}, t_k) + \cdots + g_{|S_\varphi|}(t_k)E_{|S_\varphi|}'(\mathbf{z}, t_k), \end{aligned} \quad (2b)$$

where E_φ' is the un-weighted error term, notably a scalar quantity readily computable throughout the search process, associated with the constraint C_φ for $\varphi = 1, 2, \ldots, \left|S_\varphi\right|$ at discrete time t_k.

Equation 2b can be re-written in the following more explicit format (the discrete time index is dropped from the notation for the sake of simplicity), which is defined to resemble the generic template given by the Liapunov function in Eq. 1a:

$$E(\mathbf{z}) = -\frac{1}{2}\sum_{\alpha=1}^{|S_\alpha|} g_\alpha \sum_{i=1}^{K}\sum_{j=1}^{K} d_{ij}^{\alpha}\delta_{ij}^{\alpha} z_i z_j - \sum_{\beta=1}^{|S_\beta|} g_\beta \sum_{i=1}^{K} \delta_i^{\beta} z_i. \quad (3)$$

Upon comparison of this generic error function given by Eq. 3 with the Liapunov function in Eq. 1a, weight and bias terms are defined in terms of constraint weighting coefficients as follows:

$$w_{ij} = \sum_{\alpha=1}^{|S_\alpha|} g_\alpha \delta_{ij}^{\alpha} d_{ij}^{\alpha} \text{ and } b_i = \sum_{\beta=1}^{|S_\beta|} g_\beta \delta_i^{\beta} \quad \text{for } i, j = 1, 2, \ldots, K. \quad (4)$$

The Eq. 4 exposes a sole and explicit dependency of weights on constraint weighing coefficients, and it may be feasible to exploit this unique relationship to facilitate adaptation indirectly, i.e., weights are adapted through constraint weighting coefficients and not directly in the weight space. An earlier attempt exactly did that by leveraging heuristics, i.e. increase the magnitude of the constraint weighting coefficient if the associated constraint is violated following a relaxation to a fixed point, to define values of constraint weighting coefficients [23]. Another prior approach derived bounds on initial values of the constraint weighting

coefficients in Eq. 4 to induce solutions of a given static optimization problem as stable equilibrium points in the state space of the Hopfield network dynamics [14,15]. It was also noted in the same study that in many cases the set of solutions tended to become a much smaller and proper subset of the set of all stable equilibrium points. This proved to be problematic since the network will most likely converge to a fixed point that is not a solution (local or otherwise) of the optimization problem under consideration. It is however possible to adapt values of the constraint weighing coefficients in a more mathematically rigorous and sound manner utilizing the procedure of gradient descent. One important aspect of the gradient descent based search is that it can be repeatedly performed until the network dynamics locate a feasible solution in an automated framework.

The main idea of adaptation is to perform gradient descent in the space of error versus constraint weighting coefficients to locate a local minimum. Consequently, the gradient descent based update rule for adaptation of the constraint weighting coefficients is formulated as follows:

$$g_\varphi\left(t_{k+1}\right) = g_\varphi\left(t_k\right) - \mu_\varphi \frac{\partial E\left(\mathbf{z}, t_k\right)}{\partial g_\varphi\left(t_k\right)}, \tag{5}$$

where μ_φ is the learning rate parameter that is a positive real number, $g_\varphi\left(t_k\right)$ is the constraint weighting coefficient associated with the constraint C_φ for $\varphi = 1, 2, \ldots, \left|S_\varphi\right|$ at discrete update time t_k, and $E\left(\mathbf{z}, t_k\right)$ is the value of error function at discrete time t_k, and t_k is the discrete time instant representing the conclusion of k-th relaxation of the network dynamics throughout an adaptation cycle.

Partial derivatives of the problem specific error function can easily be computed using Eq. 2a as follows:

$$\frac{\partial E\left(\mathbf{z}, t_k\right)}{\partial g_\varphi(t_k)} = E'_\varphi\left(\mathbf{z}, t_k\right) \quad \text{for } \varphi = 1, 2, \ldots, \left|S_\varphi\right|. \tag{6}$$

As a result, update equations for constraint weighting coefficients become

$$g_\varphi\left(t_{k+1}\right) = g_\varphi\left(t_k\right) + \mu_\varphi E'_\varphi\left(\mathbf{z}, t_k\right) \quad \text{for } \varphi = 1, 2, \ldots, \left|S_\varphi\right|. \tag{7}$$

It is relevant to note that the updates to the constraint weighting coefficients will always be non-negative and hence, will increase only the magnitude of the coefficient for the constraint that was violated [24].

A block diagram representation for the adaptation procedure is depicted in Fig. 1, where the $\mathbf{z}\left(t_k, \infty\right)$ represents the value of output vector following relaxation to a fixed point at the conclusion of the t_k-th cycle. Following initialization of the node outputs and the constraint weighting coefficients, through which initial values of the weight matrix entries are computed in accordance with the Eq. 4, the Hopfield network is let relax asynchronously to a fixed point. Upon convergence to a fixed point that is not a locally optimum solution, constraint weighting coefficients are updated through the Eq. 7. New values of weights are computed using the most recently updated values of constraint weighting coefficients. Then, the Hopfield network uses these new weight values to launch a new relaxation period, and the search goes on until the first local optimum solution is found.

The proposed methodology herein to adapt the constraint weighting coefficients aims to address a shortcoming of the Hopfield optimization algorithm. This shortcoming is associated with the initialization/configuration phase while the Hopfield algorithm is being set up for the specific static optimization problem. This adaptation process does not impact the gradient search process implemented by the Hopfield network which is expected to compute locally optimum solutions. Further improving the performance so that solutions approximating the

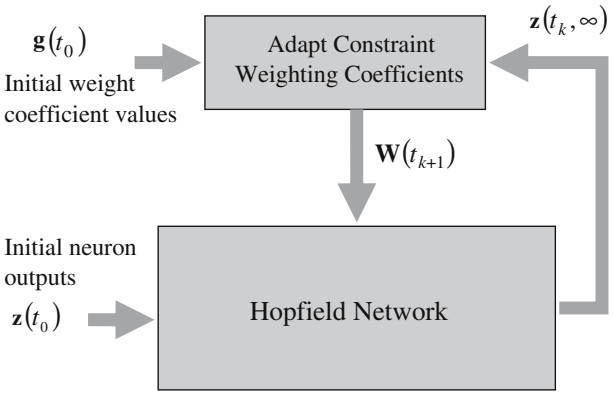

Fig. 1 Adaptation in constraint weighting coefficient space

global optima are computed will require enhancements to the search process itself: these could be in the form of stochastic (simulated annealing, mean field annealing, etc.) or chaotic mechanisms being incorporated into the search process.

2.3 Configuration of Hopfield Network for Static Optimization

Solution of static optimization problems through the Hopfield network in most cases requires a two-dimensional node array, i.e. searching for an optimal path in a graph with N vertices would require N^2 nodes arranged in a two-dimensional $N \times N$ topology. The study presented in this paper will leverage the well-known NP-complete benchmark problem, the Traveling Salesman, from the combinatorial optimization domain. Given a list of N cities, the Traveling Salesman problem (TSP) requires a visit to each city once while minimizing the total travel distance. There are two problem constraints to satisfy: each city must be visited once, and only once, and the total travel distance should be minimum. An $N \times N$ node array may be employed as the network topology for this problem, where rows and columns represent the cities and the visiting order, respectively. In this topology, any permutation matrix, i.e. each column and row has exactly one node active, is a valid solution although most likely a locally optimal one.

Consider the following error function proposed by Hopfield to map the TSP to the network topology [3],

$$E(\mathbf{z}) = -\frac{1}{2} g_{row} \sum_{i=1}^{N\times N} \sum_{j=1}^{N\times N} \delta_{ij}^{row} z_i z_j - \frac{1}{2} g_{col} \sum_{i=1}^{N\times N} \sum_{j=1}^{N\times N} \delta_{ij}^{col} z_i z_j - \frac{1}{2} g_{glo} \left(\sum_{i=1}^{N\times N} z_i - N \right)^2 - \frac{1}{2} g_{dis} \sum_{i=1}^{N\times N} \sum_{j=1}^{N\times N} d_{ij} \delta_{ij}^{dis} z_i z_j, \tag{8}$$

where

$\delta_{ij}^{row} = 1$ if $row(i) = row(j)$ or $\delta_{ij}^{row} = 0$, otherwise;
$\delta_{ij}^{col} = 1$ if $col(i) = col(j)$ or $\delta_{ij}^{col} = 0$, otherwise;
$\delta_{ij}^{dis} = 1$ if $\{|col(i) - col(j)| = 1 \wedge [row(i) \neq row(j)]\}$ or $\delta_{ij}^{dis} = 0$, otherwise;

$i \neq j;\ g_{row}, g_{col}, g_{dis}, g_{glo} \in R^{-},\ d_{ij}$ is the distance between cities $row(i)$ and $row(j)$, and superscripts/subscripts *row*, *col*, *glo*, and *dis* stand for the so-called row, column, global, and distance (inhibitory) constraints, respectively. Functions $row(i)$ and $row(j)$ return the row and column location of nodes indexed by i and j, respectively. In Eq. 8, the first double-sum term ensures that there is no more than one active neuron per row while the second double-sum does the same for each column. The third error term encourages exactly N neurons to be active for the $N \times N$ topology. The fourth term, a double sum, facilitates those solutions with smaller distances to be preferred by the search algorithm.

Comparison of this error function in Eq. 8 with the generic Liapunov function of Eq. 1a yields following definitions for the weight matrix entries and the external bias terms:

$$\begin{aligned} w_{ij} &= g_{row}\delta_{ij}^{row} + g_{col}\delta_{ij}^{col} + g_{dis}\delta_{ij}^{dis} d_{ij} + g_{glo} \text{ and} \\ b_i &= \frac{1}{2}(1-2N)\, g_{glo} \text{ for } \quad i, j = 1, 2, \ldots, N \times N; i \neq j, \end{aligned} \tag{9}$$

where note that $S_\varphi = \{C_{row}, C_{col}, C_{dis}, C_{glo}\}$, $S_\alpha = \{C_{row}, C_{col}, C_{dis}\}$ and $S_\beta = \{C_{glo}\}$. Equation 9 exposes the explicit relationship between the network weights and the constraint weighting coefficients: it facilitates computation of weight values as a function of only the constraint weighting coefficients. This observation forms the adequate basis for learning the weights through constraint weighting coefficients since adaptation can be implemented in the constraint weighting coefficient space, which is relatively very low dimensional (in general), and weights may be recomputed each time these coefficients are updated.

3 Simulation Study

The simulation study aims to assess feasibility, performance, and scalability (i.e. utility as the problem size increases) of the adaptation scheme for the Hopfield network dynamics configured for static optimization. Simulation of Hopfield neural network dynamics requires derivation of discrete-time equations using the continuous dynamics given in Eq. 1b. The specific form of discrete-time equation for the neuron dynamics will depend on the numerical integration method chosen. In this study, the discrete time equation, in one of its simpler forms, is chosen in the following construction:

$$\begin{aligned} u_i(t_{k+1}) &= \sum_{j=1}^{N\times N} w_{ij} z_j(t_k) + b_i \text{ and} \\ z_i(t_{k+1}) &= f\left[u_i(t_{k+1})\right] \quad \text{for } i = 1, 2, \ldots, N \times N \end{aligned} \tag{10}$$

where $N \times N$ is the number of neurons in the Hopfield network, $z_i(t_k)$ and $u_i(t_k)$ are the values of i-th neuron output and activation, respectively, at discrete time t_k, w_{ij} is the weight between neurons z_i and z_j subject to $w_{ij} = w_{ji}$ and $w_{ii} = 0$, b_i is the external bias input for node z_i, and $f(\cdot)$ is the activation function. Given that the application problem, the TSP, is from the domain of combinatorial optimization, a unipolar binary-valued activation function is adopted for computational cost concerns among others as it is much less costly to simulate a binary threshold function versus a continuous sigmoid function.

In addressing static optimization problems, there are two operational modes for the Hopfield network: adaptation (training or learning) and relaxation. Often these two modes are coupled. Starting with initial conditions, network dynamics are relaxed to a fixed point, which is not very likely to be a local minimum solution at least during initial trials. If the fixed point

is not a solution, the network dynamics are subjected to adaptation (i.e. adjustable parameters re-computed). Consequently, a new relaxation cycle is initiated, which is followed again by an adaptation cycle if, in fact, the fixed point is not a locally optimal solution this time around either, and so on.

Noting Eq. 2a, the process of mapping a given optimization problem to the two-dimensional Hopfield network topology suggests if a constraint weighting coefficient is a positive or negative real number, although the magnitude will not be known. Bounds can be established on magnitudes to induce solutions as local minima often at the expense of many more superfluous ones [15]. However, to assess the true inherent potential of the adaptation algorithm, magnitudes of constraint weighting coefficients will initially be set to random values in the interval [0.0,1.0]. Initial values of weights are then determined using initial values of constraint weighting coefficients through the Eq. 9. Initial values of neuron outputs are typically randomly specified as is the update order for neurons during the course of relaxation. Asynchronous update of neuron outputs, i.e. one neuron at a time, is desirable, and adopted in this study, for convergence to a fixed point rather than a limit cycle. The latter becomes a possibility for the case of synchronous update, i.e. multiple neurons updated at any given time.

3.1 Adaptation Procedure

Adaptation in constraint weighting coefficient space is implemented through the basic gradient descent approach articulated in Eqs. 5–8. The pseudocode pertaining to specifics of implementation of adaptation is further detailed in Table 1.

3.2 Managing Computational Complexity of Simulation

Simulating the Hopfield network algorithm for large instances of optimization problems poses overwhelming challenges in terms of the memory space that must be allocated for the weight matrix, which is the highest-cost data structure. In general, it is well known that the number of memory bytes required is on the order of $O\left(N^4\right)$ for N-vertex graph search problems mapped to a Hopfield network topology. Searching a 1000-vertex graph would require approximately 4×E+12 bytes of main memory storage to maintain weight matrix entries: two assumptions prevail for this computation, which are that each weight matrix entry is stored in a float type variable and a float type variable requires 4 bytes of storage space. Reasoning along the same lines, a computing platform with on the order of a couple Giga bytes of main memory, say a Windows™ XP machine, could accommodate up to 200-vertex graph search problems, most likely at the expense of an excessive number of page faults. Accordingly, we adopted the on-demand computation of weights (i.e. no weight matrix data structure is created) for large-scale TSP instances, in this case more than 100 cities, in order to circumvent the space-cost barrier posed by the creation of a weight matrix data structure [22]. This unavoidable decision, reached at as a consequence of the context of computational resources available to us, has severely limited our ability to perform as many large-scale simulations as would be desirable.

3.3 Simulation Results

Assessing the performance of the adaptation scheme with respect to quality of solutions, computational cost, and scalability (with respect to the applicability of adaptation procedure for large city counts) is a fundamental objective of the simulation study. Measuring the computational cost will be facilitated through the total number of network updates, i.e. one

Table 1 Pseudocode for Adaptation of Hopfield Dynamics in Constraint Weighting Coefficient Space

Initialization

Initialize Hopfield network constraint weighting coefficients randomly in the range [−1.0,1.0].

Initialize learning rates for constraint weighting coefficients; a preliminary empirical search may be needed for a good set of values.

Initialize weights of the Hopfield network through Eq. 4.

Initialize Hopfield network neuron outputs randomly in the range [0.0,1.0].

Adaptive Search

Relaxation

Relax Hopfield network dynamics through asynchronous update until convergence to a fixed point.

Adaptation

Update constraint weighting coefficients as in Eq. 7.

Compute constraint specific unweighted error values:

Initially let $E'_{row} = E'_{col} = E'_{dis} = E'_{glo} = 0.0$

If $\sum_{i=1}^{N} z_{row,i} > 1.0$, then $E'_{row} = E'_{row} + \sum_{i=1}^{N} z_{row,i} - 1.0$ for $row = 1, 2, \ldots, N$

If $\sum_{i=1}^{N} z_{i,col} > 1.0$, then $E'_{col} = E'_{col} + \sum_{i=1}^{N} z_{i,col} - 1.0$ for $col = 1, 2, \ldots, N$

$$E'_{dis} = \sum_{i=1}^{N \times N} \sum_{j=1}^{N \times N} d_{ij} \delta_{ij}^{dis} z_i z_j$$

$$E'_{glo} = \sum_{i=1}^{N \times N} z_i - N$$

Update learning rates for constraint weighting coefficients using the heuristic global adaptation rule [Cichocki et al., 1993]:

$$\mu_\varphi = \begin{cases} 1.05 \times \mu_\varphi, & \text{if } E'_\varphi\left(t_{k+1}\right) < E'_\varphi\left(t_k\right) \\ 0.7 \times \mu_\varphi, & \text{if } E'_\varphi\left(t_{k+1}\right) \geq 1.04 \times E'_\varphi(t_k), \quad \text{for } \varphi = row,\ col,\ dis, \text{ and } glo. \\ \mu_\varphi, & \text{otherwise} \end{cases}$$

Update constraint weighting coefficients:

$g_{row} = g_{row} - \mu_{row} E'_{row}$

$g_{col} = g_{col} - \mu_{col} E'_{col}$

$g_{dis} = g_{dis} - \mu_{dis} E'_{dis}$

If $E'_{glo} > 0.0$, then $g_{glo} = g_{glo} = \mu_{glo} E'_{glo}$

If $E'_{glo} < 0.0$, then $b = b + \mu_{glo} E'_{glo}$

Recompute weights through Eq. 4.

Termination

If termination criteria not satisfied (i.e. a local minimum solution not found or maximum number of adaptations trials attempted), continue with Adaptive Search.

complete update of outputs for all the neurons in the network. The normalized total distance (NTD), i.e. the total travel distance of a local minimum solution divided by the number of cities, will be leveraged to determine the quality of solutions.

An empirical approach is needed to initialize the values of learning rate parameters, one per constraint, appearing in Eq. 7. In a general sense, it is reasonable to start with independently determined values for these parameters. Since the TSP has four constraints, (row,

Table 2 Performance for best performing learning rate initial settings for 100-city TSP

	Mean	SD
Relax & Adapt Cycles	7.8	2.3
Network updates	27.0	17.5
Quality of solution	0.286	0.136

Table 3 Performance for the 150-city TSP

	Trial 1	Trial 2	Trial 3	Trial 4	Trial 5	Mean
Relax & Adapt Cycles	14	23	23	30	50	28
Network updates	92	102	96	206	175	134
Quality of solution	0.1300	0.4954	0.5259	0.5079	0.4440	0.4206

column, distance, and global) there are four parameter values to be initialized following an empirical process of search. The effect of these four learning rates on the adaptation algorithm performance was probed through judicious sampling of the four-dimensional parameter space. Eight cases, where each one represents a unique combination of values for the initial values of four learning rates, were simulated for the 100-city TSP instance: each case was simulated ten times. Significant performance differentiation was observed across these eight cases. One particular combination of settings, namely $\mu_{row} = 1.0$, $\mu_{col} = 1.0$, $\mu_{dis} = 0.01$, and $\mu_{glo} = \mu_{bia} = 1.0$, resulted in better quality solutions with lower computational cost compared to all the rest considered. It is also relevant to note that, regardless of the learning rate parameter settings, the adaptive Hopfield network was able to converge to a solution of the 100-city TSP in all 80 simulation cases. Statistical measures of quantities of interest are shown in Table 2 for the best performing learning rate settings. The "Quality of Solution" is measured by the normalized total distance (NTD) (i.e. total solution distance is divided by the number of cities) of the solution found, while noting that the expected value of NTD for a randomly determined solution is 0.50. The "Relax & Adapt Cycles" indicates how many times the Hopfield network converged to a fixed point and consequently went through an adaptation process, while the "Network Updates" shows the number of times a complete update of all the neurons in the network took place.

In order to test the performance of the adaptation procedure for larger problem instances, the TSP with city counts of 150, 200, 250, and 500 were studied using the learning rate initial settings adopted above and results are presented in Tables 3–7. The adaptation scheme was able to guide the network towards a local optimum solution in most cases although there were cases where search attempts proved to be futile even after substantial computational effort was expended. This happened more frequently as the scale of the problem increased and it is most likely due to excessive amount of computation needed for on-demand computation of weights. Attempts to locate a local minimum solution for city counts above 500 were not successful within a four-week maximum time allotment imposed by the resources available at the time.

Since the Hopfield network is a gradient descent based search algorithm, its performance is expected to project the typical profile of a deterministic search. Results in Tables 2–7 indicate that the network more often than not located an average quality solution although, at times, some solutions had surprisingly high quality, i.e. very low normalized

Table 4 Performance for the 200-city TSP

	Trial 1	Trial 2	Trial 3	Trial 4	Trial 5	Mean
Relax & Adapt Cycles	7	19	33	7	19	17
Network updates	73	45	104	40	107	74
Quality of solution	0.1241	0.5278	0.4999	0.1136	0.4644	0.3459

Table 5 Performance for the 250-city TSP

	Trial 1	Trial 2	Trial 3	Trial 4	Trial 5	Mean
Relax & Adapt Cycles	37	7	8	19	37	22
Network updates	135	21	36	183	253	126
Quality of solution	0.5487	0.1266	0.1069	0.5362	0.5297	0.3696

Table 6 Performance for the 400-city TSP

	Trial 1	Trial 2	Trial 3	Trial 4	Trial 5	Mean
Relax & Adapt Cycles	19	17	18	18	18	18
Network updates	267	86	117	56	266	158
Quality of solution	0.4596	0.4776	0.4630	0.4996	0.4399	0.4679

Table 7 Performance for the 500-city TSP

	Trial 1	Trial 2	Trial 3	Mean
Relax & Adapt Cycles	25	18	25	23
Network updates	261	329	333	308
Quality of solution	0.4823	0.4836	0.4867	0.4842

travel distance. Solution quality aspect of the performance appeared to scale reasonably well with the increase in the size of the problem up to 250 cities: two out of five solutions had very low NTD values for the 250-city TSP instance and average quality of solutions remained less than 0.40 across this range of city counts. However, as the problem size increases to 500 cities, performance of the adaptive Hopfield network approaches to that of a random search algorithm, although seemingly maintaining a slight edge over it. However, further validation of this observation would be justified given the limited nature of the empirical data available in this respect. The computational cost increases with the raise in the size of the problem although not in a dramatic manner: the trend of increase in the number of total relaxations is observable in Tables 5–7. One is tempted to interpret these numbers further: considering that relaxation cycle (an update) for hardware realization would typically require on the order of milliseconds or less, an update count of 1,000 would essentially require no more than one second to perform the search for a local optimum solution. This observation suggests there is promise in applying the Hopfield network as an optimization algorithm in a real-time context.

Simulation results confirm that the proposed adaptation methodology facilitates an automated procedure to determine values for constraint weighting coefficients such that the Hopfield network is consequently able to locate a locally optimum solution for the TSP. Empirical results further demonstrate that the methodology appears to scale to larger problem sizes: locally optimum solutions were computed for 500-city TSP instances within reasonable computation cost bounds although further study for even larger problem sizes would be justified.

3.4 Observations on and the Impact of the Proposed Adaptation Methodology

The proposed approach is applicable for any static optimization problem, combinatorial or otherwise, that can be mapped to the Hopfield network through a quadratic penalty/performance function. In other words, the proposed algorithm is poised to be able to search for a set of values for the penalty coefficients (e.g. constraint weighting coefficients) so that the fixed point the Hopfield network converges following relaxation of its dynamics is a locally optimum solution of the specific optimization problem under investigation. The amount of computational effort and the duration of such a search are likely to heavily depend on the nature of the optimization problem being addressed and will require a comprehensive follow-up empirical study to assess, which admittedly is not contained within the scope of work presented here.

The process of solving a static optimization problem with the Hopfield network or its derivatives requires formulation of a problem-specific penalty/performance function and empirical exploration of a suitable set of values for the penalty (constraint weighting) coefficients. The designer might have to perform a trial-and-error search to explore the penalty coefficient space to determine appropriate values for these parameters since randomly determined values will often not suffice. The work proposed herein in effect automates this very same process of trial-and-error (which might require a certain degree of expertise as well) and makes it possible to perform the same work without any noteworthy level of expertise. Accordingly, the proposed methodology does not induce additional overhead for the overall optimization process when considered from start to finish.

The basic Hopfield network implements a gradient-descent search algorithm, which has the fundamental premise of computing a locally optimal solution for a static optimization problem. The enhancement being suggested herein does not change this fact since it is related to the setup phase of the algorithm. In other words, the proposed contribution addresses a shortcoming of the algorithm associated the initialization/configuration phase and not the search phase. Accordingly, even after incorporating the enhancement being proposed, the Hopfield network performs a gradient descent search for solutions of the optimization problem. Ordinarily a relatively straightforward enhancement of the basic Hopfield algorithm along the lines of incorporating a stochastic search component, i.e. as in the mean field annealing or the Boltzmann machine, is likely to improve the solution quality dramatically at the expense of greater computational cost and partial loss of parallelism, while retaining the applicability of the proposed adaptation scheme.

The proposed methodology for adaptation of constraint weighting coefficients addresses an outstanding shortcoming associated with solving static optimization problems with the Hopfield networks. Prior to the research findings reported herein, values of constraint weighting coefficients could only be defined through empirical means or pre-computed, both of which projected their own limitations as well. This study facilitates values of penalty or constraint weighting coefficients to be defined through a sound mathematical and adaptive methodology within an automated framework as its significant contribution.

4 Conclusions

This study demonstrates feasibility and applicability of an adaptation methodology for determining values of constraint weighing coefficients for a Hopfield network configured for static optimization. The proposed adaptation scheme effectively eliminates the guesswork associated with determining appropriate values for the said parameters. The work presented herein makes an important contribution towards enhancing and effectuating the Hopfield network as a more viable algorithm for addressing optimization problems. Given the significant computational promise of Hopfield networks as an optimization algorithm for real time environments, assessing the scaling properties of the adaptation scheme for even larger scale optimization problem instances is an important extension of this work. Another direction for further research is to evaluate the utility of a stochastic enhancement of the adaptive Hopfield search algorithm for large-scale problem instances for computation of near-optimal solutions.

References

1. Smith K (1999) Neural networks for combinatorial optimization: a review of more than a decade of research. INFORMS J Comput 11(1):15–34
2. Cichocki A, Unbehauen R (1993) Neural networks for optimization and signal processing. John Wiley & B. G. Teubner, Stuttgart
3. Hopfield JJ, Tank DW (1985) Neural computation of decision in optimization problems. Biol Cyber 52:141–152
4. Atencia M, Joya G (2005) Dynamical analysis of continuous higher-order Hopfield networks for combinatorial optimization. Neural Comput 17:1802–1819
5. Atencia M, Joya G, Sandoval F (2005) Two or three things that we (intend to) know about Hopfield and tank networks. European Symposium on Artificial Neural Networks, Bruges, Belgium, pp 25–36
6. Bournez O, Campagnolo ML (2006) A survey on continuous time computation. In: Cooper B (ed) New computational paradigms. Springer
7. Smith KA, Potvin J-Y, Kwok T (2002) Neural network models for combinatorial optimization: deterministic, stochastic and chaotic approaches. Control Cyber 31(2):183–216
8. Sima J, Orponen P (2003) Continuous-time symmetric Hopfield nets are computationally universal. Neural Comput 15(3):693–733
9. Sima J, Orponen P (2003) General-purpose computation with neural networks: a survey of complexity theoretic results. Neural Comput 15(12):2727–2778
10. Sima J (2003) Energy-based computation with symmetric Hopfield nets. In: Ablameyko S, Gori M, Goras L, Piuri V (eds) Limitations and future trends in neural computation. NATO Science Series: Computer & Systems Sciences, vol 186, IOS Press, Amsterdam, 2003, pp 45–69
11. Abe S (1989) Theories on the Hopfield neural networks. In: Proceedings of the IEE international joint conference on neural networks, IEEE, Washington, DC, vol 1, pp 557–564
12. Abe S (1993) Global convergence and suppression of spurious states of the Hopfield neural networks. IEEE Trans Circ Syst-I 40(4):246–257
13. Abe S (1996) Convergence acceleration of the Hopfield neural network by optimizing integration step sizes. IEEE Trans Syst Man Cyber B 26(1):194–201
14. Serpen G, Livingston DL (2000) Determination of weights for relaxation recurrent neural networks. Neurocomputing 34:145–168
15. Serpen G, Parvin A (1997) On the performance of Hopfield network for graph search problem. Neurocomputing 14:365–381
16. Serpen G (2003) Adaptive Hopfield network. In: Proceedings of international conference on artificial neural networks. Lecture Notes in Compter Science, Springer-Verlag, Istanbul, Turkey, pp 1–7, 2003
17. Moody JE (1992) The effective number of parameters: an analysis of generalization and regularization in nonlinear learning systems. In: Moody JE, Hanson SJ, Lippman RP (eds) Advances in neural information processing systems. Morgan Kaufmann Publishers, San Mateo
18. Nowlan SJ, Hinton GE (1992) Simplifying neural networks by soft weight-sharing. Neural Comput 4:473–493

19. Lauthrup B, Hansen LK, Law I, Morch N, Svarer C, Strother SC (1994) Massive weight sharing: a cure for extremely ill-posed problems. Workshop on Supercomputing in Brain Research: From Tomography to Neural Networks, HLRZ, Julich, Germany, November 21–23, 1994
20. Hinton GE, van Camp D (1993) Keeping neural networks simple by minimizing the description length of the weights. In: Proceedings of the sixth annual ACM conference on computational learning theory (COLT 1993), July 26–28, 1993, Santa Cruz, CA, USA
21. Gallagher MR (1999) Multilayer perceptron error surfaces: visualization, structure and modelling. PhD Thesis, Department of Computer Science and Electrical Engineering, University of Queensland, Australia
22. Serpen G (2004) Managing spatio-temporal complexity in Hopfield neural network simulations for large-scale static optimization. Math Comput Simulation 64:279–293
23. Serpen G, Livingston DL (1990) An adaptive constraint satisfaction network. In: Proceedings of ASIL-OMAR conference on signals, systems and circuits, Monterey, California, pp 163–167, November 1990
24. Serpen G (2005) A heuristic and its mathematical analogue within artificial neural network adaptation context. Neural Netw World 15:129–136