# On the Capacity of Hopfield Neural Networks as EDAs for Solving Combinatorial Optimisation Problems

Kevin Swingler
*Computing Science and Maths, University of Stirling, Stirling, FK9 4LA. Scotland*

Keywords: Optimisation, Hopfield Neural Networks, Estimation of Distribution Algorithms

Abstract: Multi-modal optimisation problems are characterised by the presence of either local sub-optimal points or a number of equally optimal points. These local optima can be considered as point attractors for hill climbing search algorithms. It is desirable to be able to model them either to avoid mistaking a local optimum for a global one or to allow the discovery of multiple equally optimal solutions. Hopfield neural networks are capable of modelling a number of patterns as point attractors which are learned from known patterns. This paper shows how a Hopfield network can model a number of point attractors based on non-optimal samples from an objective function. The resulting network is shown to be able to model and generate a number of local optimal solutions up to a certain capacity. This capacity, and a method for extending it is studied.

## 1 Introduction

Optimisation based on an objective function (OF) involves finding an input to the OF that maximises its output[1]. In many cases the OF has a structure that can be exploited to speed a search considerably. There are many algorithms that guide a search by sampling from an objective function. Estimation of Distribution Algorithms (EDAs) (Mühlenbein and Paaß, 1996), (Shakya et al., 2012) build a model of the probability of sub-patterns appearing in a good solution, and then generate new solutions by sampling based on those probabilities. Objective function models (Jin, 2005) estimate the OF by fitting a model to selected samples and then search the model. Population based methods such as Genetic Algorithms (Goldberg, 1989), (De Jong, 2006) and Particle Swarm Optimisation (Kennedy, 2001) do not build an explicit model, but maintain a population of high scoring points to guide the sampling of new points.

Any OF of interest is likely to be multi-modal, which means that it contains more than one local maxima. An algorithm that guides the sampling of the OF towards high points is likely to reach a local maximum before it finds the global maximum. A local maximum is a point which has no immediate neighbours in input space with a higher output. If we assume that the OF surface is smooth (or can be smoothed) then each local maximum sits at the top of a hill delimited on all sides by local minima. A hill climb from any point on the slopes of this hill will take you to the local maximum so local maxima can be considered as attractor states for a hill climbing algorithm. One way to manage the problem of local maxima is to model these attractor states. This paper shows how a neural network can be used to explicitly model the attractor states of an OF and investigates the number of attractors a model can carry.

The motivation for using a Hopfield network is that it can model multiple point attractors in high dimensional space. The dynamics of such networks means they are guaranteed to move from any point in input space to a point of local maximum. Hopfield networks have been thoroughly studied so if they are capable of modelling the attractors of an OF, then this large body of existing research may be immediately applied. In this paper we present evidence that Hopfield networks can learn the attractor states of arbitrary OFs from a relatively small number of function samples.

**Paper Structure** This paper first describes the concepts of objective functions and attractors in objective function space. Section 2 describes the standard Hopfield neural network and section 3 introduces the Hopfield EDA (HEDA) and addresses its capacity to store local maxima as attractor states. Sections 4 and 5 describe how to train and search a HEDA using two different learning algorithms. Section 6 shows the results of some experiments investigating the capacity of different HEDA models and section 7 offers some conclusions and future directions.

[1] Optimisation might alternatively minimise a cost function, but in this paper we maximise.

### 1.1 The Problem Space

We consider binary patterns over $\mathbf{c}$ where:

$$\mathbf{c} = c_1, \ldots, c_n \qquad c_i \in \{-1, 1\} \tag{1}$$

We denote the objective function as:

$$f(\mathbf{c}) = x \qquad x \in \mathbb{R} \tag{2}$$

Local maxima in $f(\mathbf{c})$ can be viewed as the peaks of hills with monotonically increasing slopes. Each of these hills is an attractor in the search space and a simple hill climbing search from any point on its slope will lead to the hill's attractor point. The goal is to find one or more patterns in $\mathbf{c}$ that maximise $f(\mathbf{c})$ and to avoid local maxima. We will do this by modelling the attractor states.

## 2 Hopfield Networks

Hopfield networks (Hopfield, 1982) are able to store patterns as point attractors in $n$ dimensional binary space. Traditionally, known patterns are loaded directly into the network (see the learning rule 5 below), but in this paper we investigate the use of a Hopfield network to discover point attractors by sampling from an objective function. A Hopfield network is a neural network consisting of $n$ simple connected processing units. The values the units take are represented by a vector, $\mathbf{u}$:

$$\mathbf{u} = u_1, \ldots, u_n \qquad u_i \in \{-1, 1\} \tag{3}$$

The processing units are connected by symmetric weighted connections:

$$W = [w_{ij}] \tag{4}$$

where $w_{ij}$ is the strength of the connection from unit $i$ to unit $j$. Units are not connected to themselves, i.e. $w_{ii} = 0$ and connections are symmetrical, i.e. $w_{ij} = w_{ji}$. The values of the weighted connections define the point attractors and learning in a standard Hopfield network takes place by setting the pattern to be learned using formula 6 and applying the Hebbian weight update rule:

$$w_{ij} \leftarrow w_{ij} + u_i u_j \quad \forall i \neq j \tag{5}$$

A single pattern, $\mathbf{c}$ is set by

$$\forall i \quad u_i \leftarrow c_i \tag{6}$$

where $\leftarrow$ indicates assignment. Pattern recall is performed by allowing the network to settle to an attractor state determined by the values of its weights. The unit update rule during settling is

$$a_i \leftarrow \sum w_{ji} u_j \tag{7}$$

where $a_i$ is a temporary activation value, following which the unit's value is capped by a threshold, $\theta$, such that:

$$u_i \leftarrow \begin{cases} 1 & \text{if } a_i > \theta \\ -1 & \text{otherwise} \end{cases} \tag{8}$$

In this paper, we will always use $\theta = 0$. The process of settling repeatedly uses the unit update rule of formulae 7 and 8 for a randomly selected unit in the network until no update produces a change in unit values. At that point, the network is said to have settled. The symmetrical weights and zero self-connections mean that the network is a Lyapunov function, which guarantees that the network will settle to a fixed point from any starting point.

With the above restrictions in place, the network has an energy function that determines the set of possible stable states into which it will settle. The energy function is defined as:

$$E = -\frac{1}{2} \sum_{i,j} w_{ij} u_i u_j \tag{9}$$

Settling the network, by formulae 7 and 8 produces a pattern corresponding to a local minimum of $E$ in equation 9. Hopfield networks have been used to solve optimization tasks such as the travelling salesman problem (Hopfield and Tank, 1985) but weights are set by an analysis of the problem rather than by learning. In the next section, we show how random patterns and an objective function can be used to train a Hopfield network as a search technique.

## 3 Hopfield EDAs

We define a Hopfield EDA (HEDA) as an EDA implemented by means of a Hopfield neural network. Although a traditional Hopfield network contains only second order connections, we will allow higher order networks and denote a HEDA of order $m$ by HEDA$^{\mathrm{m}}$. We will concentrate mostly on second order models based on the traditional Hopfield network - HEDA$^2$ models.

### 3.1 HEDA Capacity

The capacity of a HEDA is the number of distinct attractors it can model. As each attractor is a single local maximum, the capacity of the network determines the number of local maxima a HEDA can absorb on its way to the global maximum.

There is a limit on the number of attractor states a Hopfield network can represent. For random patterns, (McEliece et al., 1987) states that the capacity of such a network is $n/(4\ln n)$ where $n$ is the number of elements in the model.

(Storkey and Valabregue, 1999) suggest an alternative to the Hebbian learning rule that increases the capacity of a Hopfield network. This new learning rule can be used to increase the number of attractors in a HEDA$^2$ and so increase the number of local maxima it is able to model. A Hopfield network of order 2 trained with Storkey's learning rule has a capacity of $n/\sqrt{2\ln n}$.

The Hopfield approach described for HEDA$^{\mathrm{m}}$ can be extended to higher orders, with a corresponding increase in storage capacity, along with an associated exponential increase in processing time. (Kubota, 2007) states that the capacity of order $m$ associative memories is $O(n^m/\ln n)$.

# 4 Training a HEDA

In this section we describe a method for training a HEDA$^2$. The principles apply equally to HEDAs of higher order. During learning, candidate solutions are generated randomly one at a time. Each candidate solution is evaluated using the objective function and the result is used as a learning rate in the Hebbian weight update rule (see update rule 10). Consequently, each pattern is learned with a different strength, which reflects its quality as a solution.

## 4.1 The New Weight Update Rule

Hopfield networks have a limited capacity for storing patterns. If a number of patterns greater than this capacity are learned, patterns interfere with each other producing spurious states, which are a combination of more than one pattern. To learn the point attractors of local optima without ever sampling those points, we need to create spurious states that are a combination of lower points. We do this by over-filling a Hopfield network with samples and introducing a strength of learning so that higher scoring patterns contribute more to the new spurious states. This yields a simple modification to the Hebbian rule:

$$w_{ij} \leftarrow w_{ij} + f(c)u_i u_j \tag{10}$$

where $f(c)$ is the objective function. This has the effect of learning high scoring second order subpatterns more than lower scoring ones. Note that due to the symmetry of the weight connections, each attractor has an associated inverse pattern that is also an attractor. The means that both the pattern and its inverse may need to be scored to tell the solutions apart from their inverse twins.

## 4.2 The Learning Algorithm

The learning algorithm proceeds as follows:

1. Set up a Hopfield network with $W_{ij}$=0 for all $i, j$
2. Repeat the following until one or more stopping criteria are met
   (a) Generate a random pattern, $\mathbf{c}$, where each $c_i$ has an equal probability of being set to 1 or -1
   (b) Calculate $f(\mathbf{c})$ by equation 2
   (c) Load $\mathbf{c}$ using formula 6
   (d) Update the weight matrix $W$ using the learning rule in formula 10

Stop when a pattern of required quality has been found or when the attractor states become stable or the network reaches capacity.

# 5 Searching a HEDA Model

Once a number of solutions have been found, the network will have a number of local attractors. By presenting a new pattern and allowing the network to settle, the closest solution (or locally optimal solution) will be found. This settling does not require $f(\mathbf{c})$ to be evaluated. In cases where a partial solution is already known, the HEDA may be used to find the closest local optimum. Where the global optimum is required, the model needs to be searched. This is quite fast because the objective function does not have to be evaluated very often. The search could be carried out in a random fashion, picking points and settling to their attractor states until an acceptable score is produced, or using a more sophisticated search method such as simulated annealing (Hertz et al., 1991).

## 5.1 Improving Capacity

Storkey (Storkey and Valabregue, 1999) introduced a new learning rule for Hopfield networks that increased the capacity of a network compared to using the Hebbian rule. The new weight update rule is:

$$w_{ij} \leftarrow w_{ij}^{t-1} + \frac{1}{n}u_i u_j - \frac{1}{n}u_i h_{ji} - \frac{1}{n}u_j h_{ij} \tag{11}$$

where

$$h_{ij} \leftarrow \sum_{k \neq i,j} w_{ik}^{t-1} u_k \tag{12}$$

and $w_{ij}^{t-1}$ is the weight at the previous time step.

The new terms, $h_{ji}$ and $h_{ij}$ have the effect of creating a local field around $w_{ij}$ that reduces the lower order noise brought about by the interaction of different attractors.

To use this learning rule in a HEDA, we make the following alterations to the update rules:

$$w_{ij} \leftarrow w_{ij}^{t-1} + \frac{1}{n}(u_i u_j - u_i h_{ji} - u_j h_{ij}) f(p) \qquad (13)$$

and

$$h_{ij} \leftarrow \sum_{k \neq i,j} w_{ik}^{t-1} u_k \beta \qquad (14)$$

where $\beta < 1$ is a discount parameter that controls how much damping is applied to the learning rule.

# 6 Experimental Results

The following experimental results test the new learning rules. We first introduce the objective functions used and then describe the experiments.

## 6.1 Experimental Objective Functions

We use two types of objective function: a set of distinct target patterns and a concept that contains dependencies of order 2.

In the first case, the set of target patterns are denoted as the set $\mathbf{t}$:

$$\mathbf{t} = \{t_1, \ldots, t_s\} \qquad (15)$$

We then define the objective function as an inverse normalised weighted Hamming distance between $\mathbf{c}$ and each target pattern $t_j$ in $\mathbf{t}$ as.

$$f(\mathbf{c}|t_j) = 1 - \sum \frac{\delta_{c_i,t_{ji}}}{n} \qquad (16)$$

where $t_{ji}$ is element $i$ of target $j$ and $\delta_{c_j,t_{ji}}$ is the Kronecker delta function between pattern element $i$ in $t_j$ and its equivalent in $\mathbf{c}$. We take the score of a single pattern to be the maximal score of all the members of the target set.

$$f(\mathbf{c}|\mathbf{t}) = max_{j=1\ldots s}(f(\mathbf{c}|t_j)) \qquad (17)$$

## 6.2 Testing HEDA$^2$ Capacity by Learning and Searching

This set of experiments compares the capacity of a normally trained Hopfield network with the search capacity of a HEDA$^2$. We will compare two learning rules (Hebbian and Storkey). The experiments are repeated many times, all using randomly generated target patterns where each element has an equal probability of being +1 or -1.

### 6.2.1 Experiment 1 - Hebbian HEDA$^2$ Capacity

In experiment 1 we compare Hebbian trained Hopfield networks with their equivalent HEDA$^2$ models. The aim is to discover whether or not the HEDA$^2$ model can obtain the capacity of the Hopfield network. Hopfield networks were trained on patterns using standard Hebbian learning, with one pattern at a time being added until the network's capacity was exceeded. At this point, the learned patterns were set to be the targets for the HEDA$^2$ search and the network's weights reset.

100 repeated trials were made training HEDA$^2$ networks ranging in size from 10 to 100 units in steps of 5. For each trial, the capacity of the trained network, the number of those patterns discovered by random sampling and the time taken to find them all was recorded.

**Results** Regardless of the capacity or size of the Hopfield network, the HEDA$^2$ search was always able to discover every pattern learned during the capacity filling stage of the test. From this, we can conclude that the capacity of a HEDA$^2$ for storing local optima is the same as the capacity for the equivalent Hopfield network when searching for a set of random targets.

Figure 1 shows the relationship between Hopfield network size and capacity. The spread of capacity values is wide - varying with the level of interdependence between the random patterns. The chart shows the range and the inter-quartile range of capacity for each network size.

Capacity is linear with network size in terms of units, but the number of weights in a network increases quadratically with capacity.

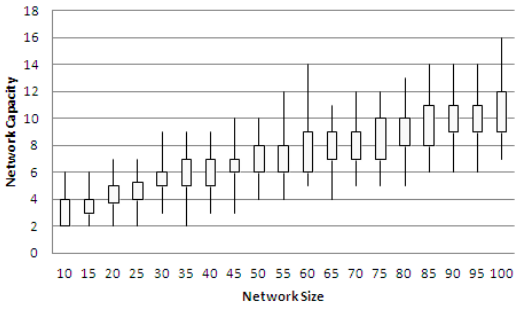


Figure 1: The learned capacity of HEDA$^2$ models of various sizes across 100 trials.

Figure 2 shows the number of samples required to find all patterns against network capacity. By curve fitting to the data shown, we find that the number of samples required to find all solutions is quadratic with number of solutions.

In fact, as the search space grows, the number of iterations required to fully characterise the search space, as a proportion of the size of search space diminishes exponentially. For networks of size 100, the search space has $2^{100}$ possible states and the HEDA$^2$ is able to find all of the targets in an average of around 355,000 samples. That is a sample consisting of $2.8^{-25}$ of all possible patterns. In this modelling exercise, no evolution towards a target took place; the learning took place using random sampling.

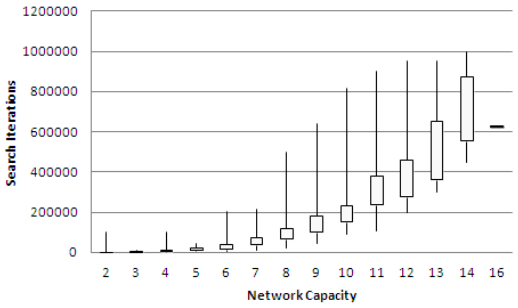


Figure 2: The number of trials needed to find all local optima in a Hopfield network filled to capacity, plotted against the number of patterns to find.

### 6.2.2 Experiment 2 - Storkey HEDA$^2$ Capacity

In experiment 2 we repeat experiment 1 but use the Storkey learning rule rather than the Hebbian version. The experimental procedure is the same as that described above, except that we only have samples from networks up to size 60, as the process for larger networks is very slow.

**Results** As with the Hebbian learning, Storkey trained HEDA$^2$ search was always able to discover every pattern learned during the capacity filling stage of the test. This shows that the improved learning rule will deliver the increased capacity for capturing local optima that we sought. The cost of this capacity is a far slower learning algorithm, however.

Figure 3 shows the relationship between Hopfield network size and capacity for the Storkey trained network.

Figure 4 shows the number of samples required to find all patterns against network size when using the Storkey rule. Again, we see that search iterations increase quadratically with network capacity.

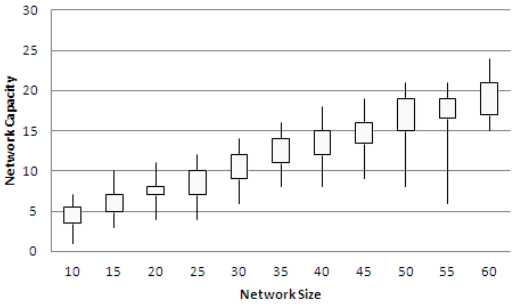


Figure 3: The learned capacity of Storkey trained HEDA$^2$ models.

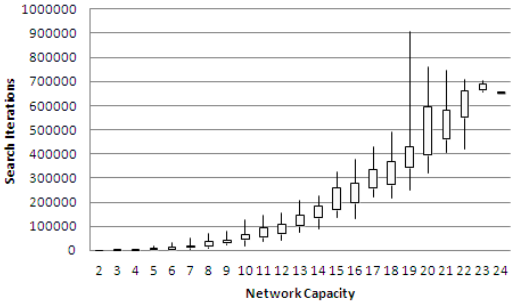


Figure 4: The number of trials needed to find all local optima in a Hopfield network filled to capacity using the Storkey learning rule, plotted against the number of patterns to find.

## 6.3 Experiments with Second Order Patterns

The concept of vertical symmetry in a pattern involves a second order rule. It is not possible to score the contribution of pattern elements in isolation - each one must be considered with its mirror partner. By designing an objective function that scores a pattern in terms of its symmetry, we show how a HEDA$^2$ can learn a second order concept and so generate many patterns that score perfectly against this concept, despite having never seen any such patterns during training.

Patterns were generated in a 6 x 6 image of binary pixels. There are $2^{36}$ (68,719,476,736) possible patterns in such a matrix. Of those, there will be one vertically symmetrical pattern for every possible pattern in one half of the image. There are $2^{18}$ (262,144) such half patterns, representing 0.00038% of the total number of possible patterns.

In this trial the network was allowed to learn until ten different symmetrical patterns had been found. At this point, the learning process was terminated and the network was tested with a set of local searches designed to count the number of attractor states learned.

**Results** Across 50 trials, the 36 node HEDA took an average of 9932 pattern evaluations before it terminated having found 10 perfect scoring patterns. A record of the samples made showed that none of the randomly generated patterns used during training gained a perfect score, so the network was only trained on less than perfect patterns. The average trained model's capacity was found to be 132, which gives an average of 75 OF evaluations per pattern found. Figure 5 shows two examples of random starting patterns and their associated attractor states.

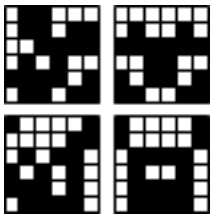

Figure 5: Two examples of fixed point attractors of the HEDA$^2$ trained on the second order concept of symmetry. The left hand column shows random starting points and the right hand column shows the associated point attractor state.

## 7 Conclusions

It is possible to adapt both the Hebbian and Storkey learning rules for Hopfield networks to allow them to learn the attractor states corresponding to multiple local maxima based on random samples from an objective function. We have experimentally shown that the capacity of these networks is at least equal to the capacity of a Hopfield network trained directly on the attractor points.

Networks trained in this way are able to find a set of attractors by undirected sampling - that is with no evolution of solutions - in a number of samples that is a very small fraction of the size of the search space.

In a second order network, as network size, $n$ varies, we have seen that search space grows exponentially with $n$, the number of local optima that can be stored grows linearly with $n$ and the time to find all local optima grows quadratically with $n$.

Future work will address a number of area including higher order networks and networks of variable order; an evolutionary approach to training the networks when the number of local optima is higher than the capacity of the network; the effects of spurious attractors; the use of the Energy function of equation 9 as a proxy for objective function evaluations; and the smoothing of noisy landscapes. Hopfield networks have been extensively studied so there is a wealth of research on which to draw during future work.

## ACKNOWLEDGEMENTS

Thank you David Cairns and Leslie Smith for your helpful comments on earlier versions of this paper.

## REFERENCES

De Jong, K. (2006). *Evolutionary computation : a unified approach*. MIT Press, Cambridge, Mass.

Goldberg, D. E. (1989). *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley Professional, 1 edition.

Hertz, J., Krogh, A., and Palmer, R. G. (1991). *Introduction to the Theory of Neural Computation*. Addison-Wesley, New York.

Hopfield, J. J. (1982). Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the National Academy of Sciences USA*, 79(8):2554–2558.

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

Jin, Y. (2005). A comprehensive survey of fitness approximation in evolutionary computation. *Soft Computing - A Fusion of Foundations, Methodologies and Applications*, 9:3–12.

Kennedy, J. (2001). *Swarm intelligence*. Morgan Kaufmann Publishers, San Francisco.

Kubota, T. (2007). A higher order associative memory with mcculloch-pitts neurons and plastic synapses. In *Neural Networks, 2007. IJCNN 2007. International Joint Conference on*, pages 1982 –1989.

McEliece, R., Posner, E., Rodemich, E., and Venkatesh, S. (1987). The capacity of the hopfield associative memory. *Information Theory, IEEE Transactions on*, 33(4):461 – 482.

Mühlenbein, H. and Paaß, G. (1996). From recombination of genes to the estimation of distributions i. binary parameters. In Voigt, H.-M., Ebeling, W., Rechenberg, I., and Schwefel, H.-P., editors, *Parallel Problem Solving from Nature PPSN IV*, volume 1141 of *Lecture Notes in Computer Science*, pages 178–187. Springer Berlin / Heidelberg.

Shakya, S., McCall, J., Brownlee, A., and Owusu, G. (2012). Deum - distribution estimation using markov networks. In Shakya, S. and Santana, R., editors, *Markov Networks in Evolutionary Computation*, volume 14 of *Adaptation, Learning, and Optimization*, pages 55–71. Springer Berlin Heidelberg.

Storkey, A. J. and Valabregue, R. (1999). The basins of attraction of a new hopfield learning rule. *Neural Netw.*, 12(6):869–876.