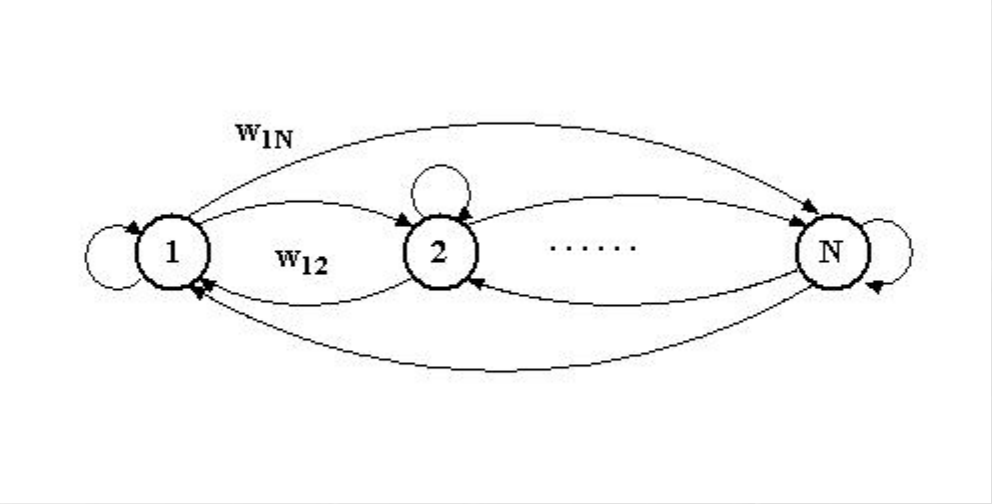
Hopfield Network

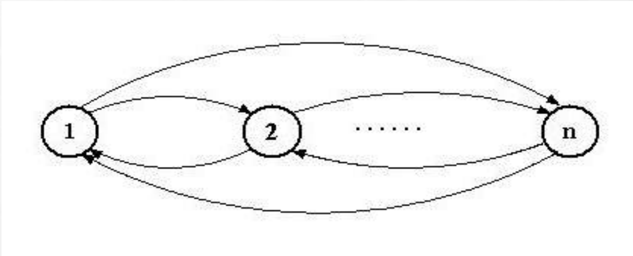
- Single Layer Recurrent Network
- Bidirectional Symmetric Connection
- Binary / Continuous Units
- Associative Memory
- Optimization Problem



Hopfield Model – Discrete Case

Recurrent neural network that uses McCulloch and Pitt's (binary) neurons.

Update rule is stochastic.



Eeach neuron has two "states" : V_i^L , V_i^H

Usually : $\begin{cases} V_i^L = -1 \ , \ V_i^H = 1 \\ V_i^L = 0 \ , \ V_i^H = 1 \end{cases}$

Input to neuron *i* is :

$$H_i = \sum_{j \neq i} w_{ij} V_j + I_i$$

Where:

- w_{ij} = strength of the connection from *j* to *i*
- V_j = state (or output) of neuron *j*
- I_i = external input to neuron *i*

Hopfield Model – Discrete Case

Each neuron updates its state in an *asynchronous* way, using the following rule:

$$V_i = \begin{cases} -1 & \text{if} \quad H_i = \sum_{j \neq i} w_{ij} V_j + I_i \; < 0 \\ +1 & \text{if} \quad H_i = \sum_{j \neq i} w_{ij} V_j + I_i \; > 0 \end{cases}$$

The updating of states is a *stochastic* process:

To select the to-be-updated neurons we can proceed in either of two ways:

- At each time step select at random a unit *i* to be updated
  (useful for simulation)

- Let each unit independently choose to update itself with
  some constant probability per unit time
  (useful for modeling and hardware implementation)

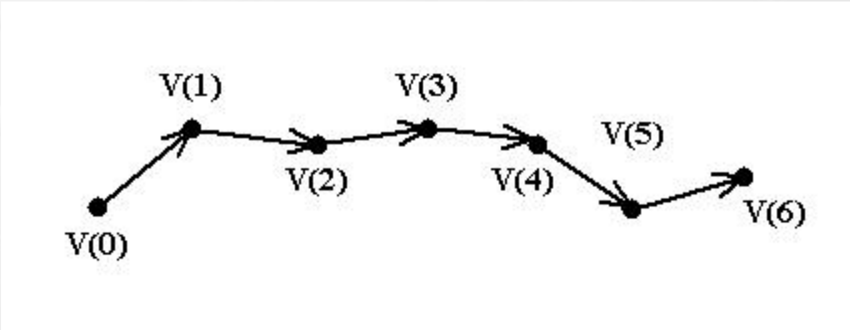
Dynamics of Hopfield Model

In contrast to feed-forward networks (wich are “static”) Hopfield networks are dynamical system.

The network starts from an initial state

$$V(0) = (V_1(0), \ldots.. ,V_n(0))^T$$

and evolves in state space following a trajectory:



Until it reaches a fixed point:

$$V(t+1) = V(t)$$

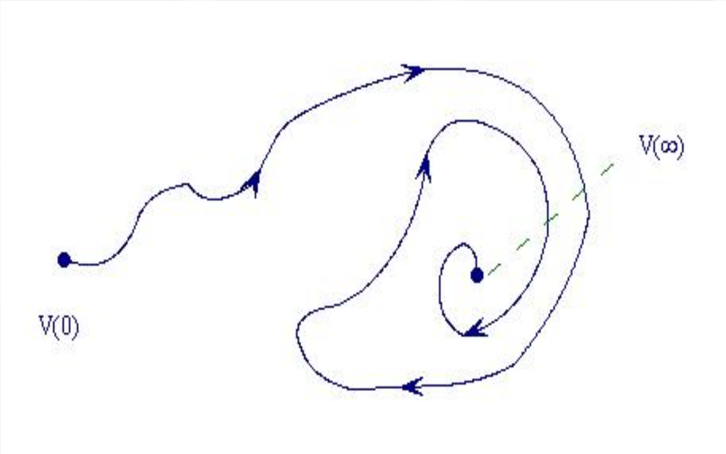
Dynamics of Hopfield Networks

What is the dynamical behavior of a Hopfield network ?

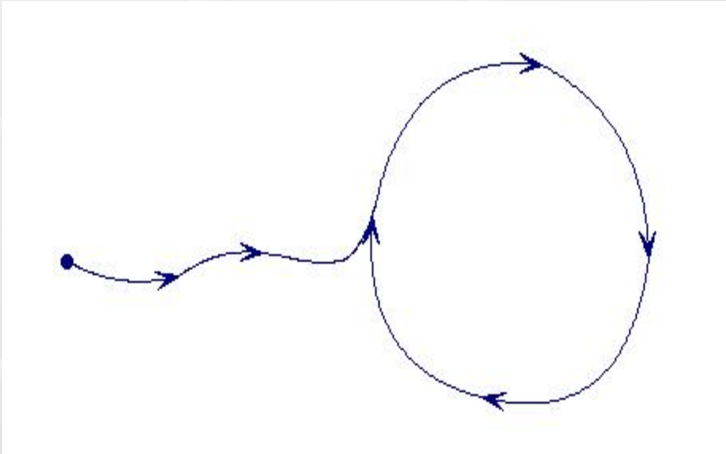
Does it coverge ?

Does it produce cycles ?

Examples



(a)



(b)

Dynamics of Hopfield Networks

To study the dynamical behavior of Hopfield networks we make the following assumption:

$$w_{ij} = w_{ji} \qquad \forall i, j = 1 \ldots n$$

In other words, if W = (w_{ij}) is the weight matrix we assume:

$$W = W^T$$

In this case the network always converges to a fixed point.
In this case the system posseses a *Liapunov* (or energy) function that is minimized as the process evolves.

The Energy Function – Discrete Case

Consider the following real function:

$$E = -\frac{1}{2}\sum_{i=1}^{n}\sum_{\substack{j=1 \\ j\neq i}}^{n} w_{ij}\, V_i\, V_j - \sum_{i=1}^{n} I_i\, V_i$$

and let $\Delta E = E(t+1) - E(t)$

Assuming that neuron *h* has changed its state, we have:

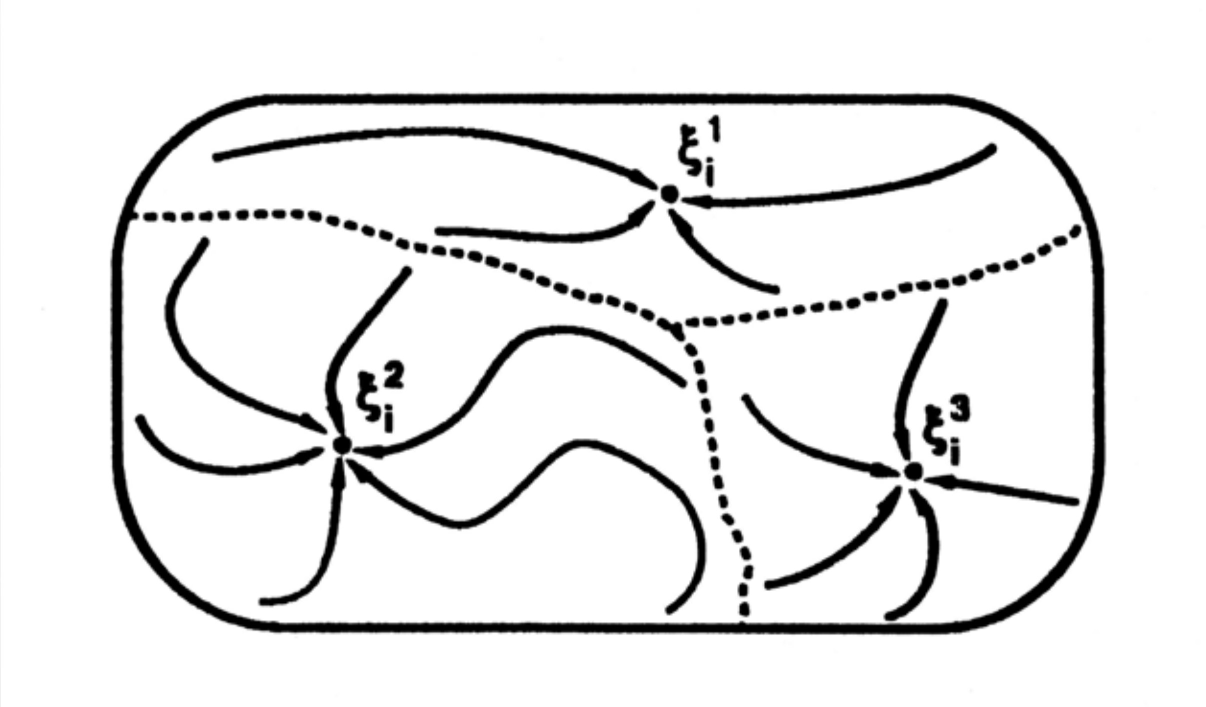
$$\Delta E = -\left[\underbrace{\sum_{j\neq h} w_{hj}\, V_j + I_h}_{H_h}\right] \Delta V_h$$

But H_h and ΔV_h have the same sign.

Hence

$$\Delta E \leq 0 \qquad \text{(provided that } W = W^T\text{)}$$

Schematic configuration space



model with three attractors

Hopfield Net As Associative Memory

Store a set of p patterns x^{μ}, $\mu = 1,\ldots,p$,in such a way that when presented with a new pattern x, the network responds by producing that stored pattern which most closely resembles x.

- N binary units, with outputs $s_1,\ldots,s_N$
- Stored patterns and test patterns are binary (0/1,±1)
- Connection weights (Hebb Rule)

  Hebb suggested changes in synaptic strengths proportional to the correlation between the firing of the pre and post-synaptic neurons.

$$w_{ij} = \frac{1}{N}\sum_{\mu=1}^{p} x_i^{\mu} x_j^{\mu}$$

- Recall mechanism

$$s_i = Sgn\left(\sum_j w_{ij}\, s_j - \theta_i\right)$$

  Synchronous / Asynchronous updating

- Pattern information is stored in equilibrium states of the network

Example With Two Patterns

- Two patterns

$X^1 = (-1,-1,-1,+1)$

$X^2 = (+1,+1,+1,+1)$

- Compute weights

$$w_{ij} = \frac{1}{4}\sum_{\mu=1}^{2} x_i^{\mu} x_j^{\mu}$$

- Weight matrix

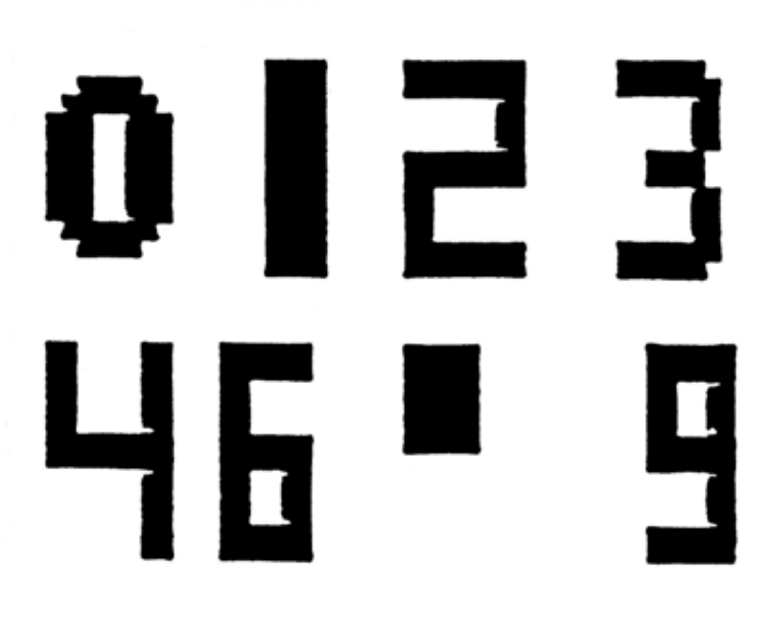
$$w = \frac{1}{4}\begin{bmatrix} 2 & 2 & 2 & 0 \\ 2 & 2 & 2 & 0 \\ 2 & 2 & 2 & 0 \\ 0 & 0 & 0 & 2 \end{bmatrix}$$

- Recall

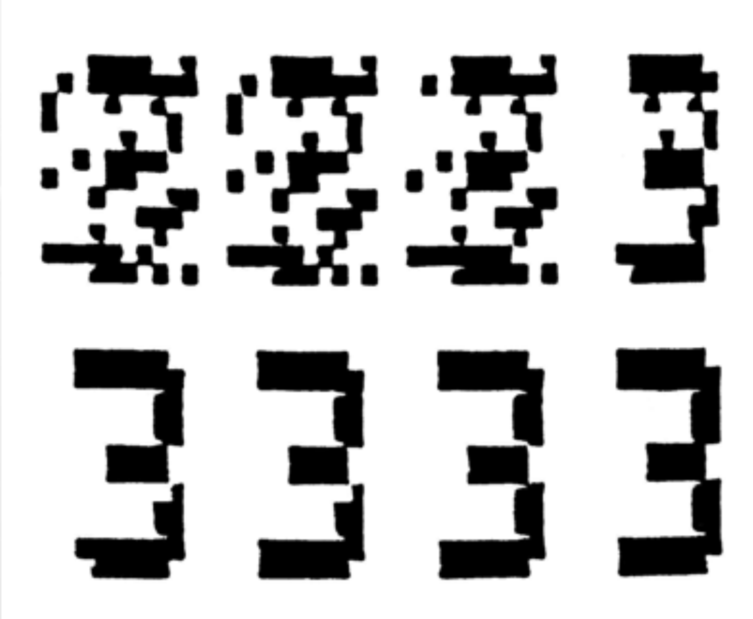
$$s_i = Sgn\left(\sum_j w_{ij}\, s_j\right)$$

- Input (-1,-1,-1,+1) → (-1,-1,-1,+1) stable
- Input (-1,+1,+1,+1) → (+1,+1,+1,+1) stable
- Input (-1,-1,-1,-1) → (-1,-1,-1,-1) spurious

Associative Memory Examples



A) EIGHT EXEMPLAR PATTERNS



B) OUTPUT PATTERNS FOR NOISY "3" INPUT

An example of the behavior of a Hopfield net when used as a content-addressable memory. A 120 node net was trained using the eight examplars shown in (A). The pattern for the digit “3” was corrupted by randomly reversing each bit with a proba-bility of 0.25 and then applied to the net at time zero.

Outputs at time zero and after the first seven iterations are shown in (B).

Associative Memory Examples

Example of how an associative memory can reconstruct images. These are binary images with 130 x 180 pixels. The images on the right were recalled by the memory after presentation of the corrupted images shown on the left. The middle column shows some intermediate states. A sparsely connected Hopfield network with seven stored images was used.

Storage Capacity of Hopfield Network

- There is a maximum limit on the number of random patterns that a Hopfield network can store

  $$P_{max} \approx 0.15N$$

  If $p < 0.15N$, almost perfect recall

- If memory patterns are orthogonal vectors instead of random patterns, then more patterns can be stored. However, this is not useful.

- Evoked memory is not necessarily the memory pattern that is most similar to the input pattern

- All patterns are not remembered with equal emphasis, some are evoked inappropriately often

- Sometimes the network evokes spurious states

Hopfield Model – Continuous Case

The Hopfield model can be generalized using continuous activation functions.

More plausible model.

In this case:

$$V_i = g_\beta(u_i) = g_\beta\left(\sum_j W_{ij} V_j + I_i\right)$$

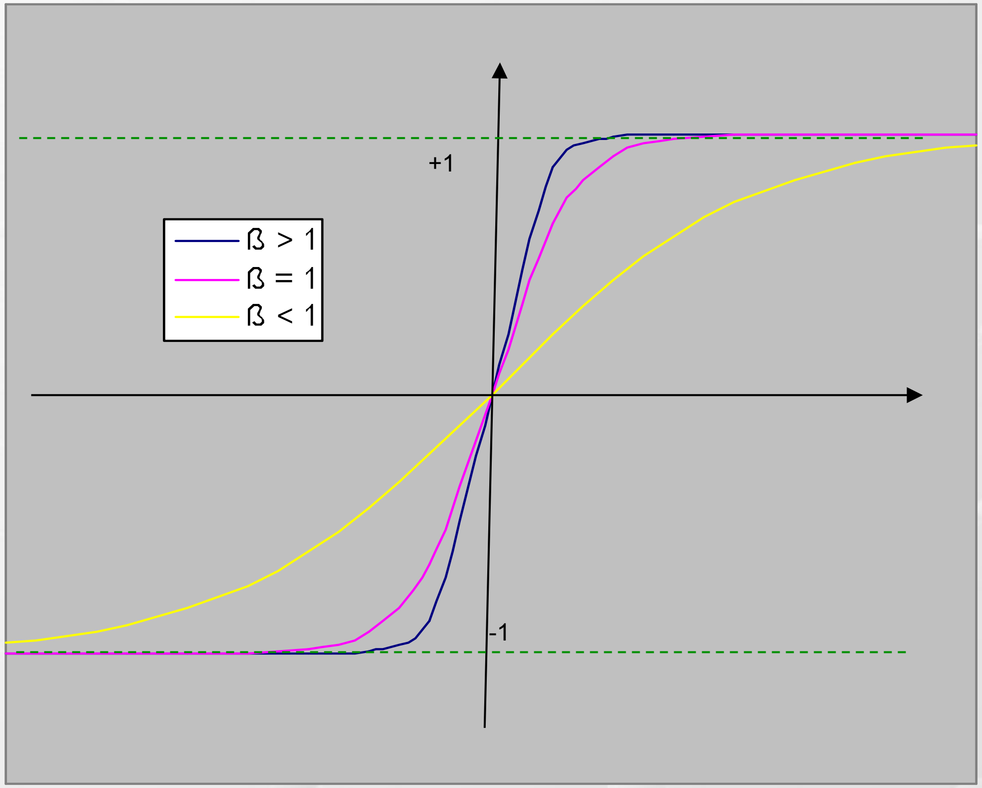
where g_β is a continuous, increasing, non linear function.

Examples

$$tanh(\beta u) = \frac{e^{\beta u} - e^{-\beta u}}{e^{\beta u} + e^{-\beta u}} \qquad \in]-1,1[$$

$$g_\beta(u) = \frac{1}{1 + e^{-2\beta u}} \qquad \in]0,1[$$

Funzione di attivazione



$$f\left(x\right) = tanh\left(\beta\, x\right)$$

Updating Rules

Several possible choices for updating the units :

Asynchronous updating: one unit at a time is selected to have its output set

Synchronous updating: at each time step all units have their output set

Continuous updating: all units continuously and simultaneously change their outputs

Continuous Hopfield Models

Using the continuous updating rule, the network evolves according to the following set of (coupled) differential equations:

$$\tau_i \frac{dV_i}{dt} = -V_i + g_\beta(u_i) = -V_i + g_\beta\left(\sum_j w_{ij} V_j + I_i\right)$$

where τ_i are suitable time constants ($\tau_i > 0$).

Note When the system reaches a fixed point ($dV_i / dt = 0 \quad \forall i$) we get

$$V_i = g_\beta(u_i)$$

Indeed, we study a very similar dynamics

$$\tau_i \frac{du_i}{dt} = -u_i + \sum_j w_{ij} g_\beta(u_j) + I_i$$

The Energy Function

As the discrete model, the continuous Hopfield network has an "energy" function, provided that $W = W^T$:

$$E = -\frac{1}{2}\sum_i \sum_j w_{ij} V_i V_j + \sum_i \int_0^{Vi} g_\beta^{-1}(V)\, dV - \sum_i I_i V_i$$

Easy to prove that

$$\frac{dE}{dt} \leq 0$$

with equality iff the net reaches a fixed point.

Modello di Hopfield continuo (energia)

$$\frac{dE}{dt} = -\frac{1}{2}\sum_{ij} w_{ij}\frac{dV_i}{dt}V_j - \frac{1}{2}\sum_{ij} w_{ij}V_i\frac{dV_j}{dt} + \sum_i g_\beta^{-1}(V_i)\frac{dV_i}{dt} - \sum_i I_i\frac{dV_i}{dt}$$

$$= -\sum_{ij} w_{ij}\frac{dV_i}{dt}V_j + \sum_i g_\beta^{-1}(V_i)\frac{dV_i}{dt} - \sum_i I_i\frac{dV_i}{dt}$$

$$= -\sum_i \frac{dV_i}{dt}\left(\sum_j w_{ij}V_j - u_i + I_i\right)$$

$$= -\sum_i \tau_i \frac{dV_i}{dt}\frac{du_i}{dt}$$

$$= -\sum_i \tau_i\, g'_\beta(u_i)\left(\frac{du_i}{dt}\right)^2 \leq 0$$

Perché g_β è monotona crescente e $\tau_i > 0$.

N.B. $$\frac{dE}{dt} = 0 \quad \Leftrightarrow \quad \frac{du_i}{dt} = 0$$

cioè u_i è un punto di equilibrio

Modello di Hopfield continuo (relazione con il modello discreto)

Esiste una relazione stretta tra il modello continuo e quello discreto.

Si noti che :

$$V_i = g_\beta(u_i) = g_1(\beta u_i) \equiv g(\beta u_i)$$

quindi :

$$u_i = \frac{1}{\beta} g^{-1}(V_i)$$

Il 2º termine in E diventa :

$$\frac{1}{\beta} \sum_i \int_0^{V_i} g^{-1}(V_i)\, dV$$

L'integrale è positivo (0 se V_i=0).

Per $\beta \to \infty$ il termine diventa trascurabile, quindi la funzione E del modello continuo diventa identica a quello del modello discreto

Optimization Using Hopfield Network

- Energy function of Hopfield network

$$E = -\frac{1}{2}\sum_i \sum_j w_{ij} V_i V_j - \sum_i I_i V_i$$

- The network will evolve into a (locally / globally) minimum energy state

- Any quadratic cost function can be rewritten as the Hopfield network Energy function. Therefore, it can be minimized using Hopfield network.

- Classical Traveling Salesperson Problem (TSP)

- Many other applications
  - 2-D, 3-D object recognition
  - Image restoration
  - Stereo matching
  - Computing optical flow

The Traveling Salesman Problem

Problem statement: A travelling salesman must visit every city in his territory exactly once and then return to his starting point. Given the cost of travel between all pairs of cities, find the minimum cost tour.

- NP-Complete Problem
- Exhaustive Enumeration:

  n nodes, $n!$ enumerations,

  $(n-1)!$ distinct enumerations

  $\frac{(n-1)!}{2}$ distinct undirected enumerations

Example:

$n = 10$, $19!/2 = 1.2 \times 10^{18}$

The Traveling Salesman Problem

TSP: find the shortest tour connecting a set of cities.

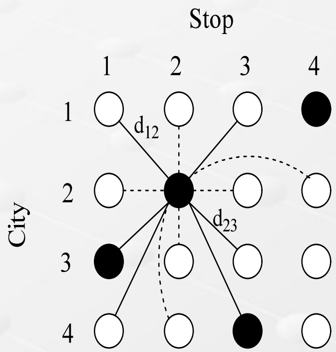
Following Hopfield & Tank (1985) a tour can be represented by a permutation matrix:

	1	2	3	4
A	0	1	0	0
B	1	0	0	0
C	0	0	1	0
D	0	0	0	1

⟷ Tour: BACD

The Traveling Salesman Problem

The TSP, showing a good (a) and a bad (b) solution to the same problem



Network to solve a four-city TSP. Solid and open circles denote units that are on and off respectively when the net is representing the tour 3-2-4-1.

The connections are shown only for unit n_{22}; solid lines are inhibitory connections of strength $-d_{ij}$, and dotted lines are uniform inhibitory connections of strength $-\gamma$. All connections are symmetric. Thresholds are not shown.

Artificial Neural Network Solution

Solution to *n*-city problem is presented in an *n* x *n* permutation matrix V

X = city

i = stop at wich the city is visited

Voltage output: $V_{X,i}$

Connection weights: $T_{Xi,Yj}$

n^2 neurons

$V_{X,i}$ = 1 if city X is visited at stop i

d_{XY} = distance between city X and city Y

Artificial Neural Network Solution

- Data term:

  We want to minimize the total distance

$$E_1 = \frac{D}{2} \sum_{X} \sum_{Y \neq X} \sum_{i} d_{XY} V_{X,i} \left(V_{Y,i+1} + V_{Y,i-1} \right)$$

- Constraint terms:

  Each city must be visited once

$$E_2 = \frac{A}{2} \sum_{X} \sum_{i} \sum_{j \neq i} V_{X,i} V_{X,j}$$

Each stop must contain one city

$$E_3 = \frac{B}{2} \sum_{i} \sum_{X} \sum_{Y \neq X} V_{X,i} V_{Y,i}$$

The matrix must contain *n* entries

$$E_4 = \frac{C}{2} \left(\sum_{X} \sum_{i} V_{X,i} - n \right)^2$$

Artificial Neural Network Solution

- A, B, C, and D are positive constants
- Indici modulo *n*

Total cost function

$$E = \frac{A}{2}\sum_{X}\sum_{i}\sum_{j\neq i} V_{X,i}V_{X,j} + \frac{B}{2}\sum_{i}\sum_{X}\sum_{Y\neq X} V_{X,i}V_{Y,i} + \frac{C}{2}\left(\sum_{X}\sum_{i} V_{X,i} - n\right)^2 + \frac{D}{2}\sum_{X}\sum_{Y\neq X}\sum_{i} d_{XY}V_{X,i}\left(V_{Y,i+1} + V_{Y,i-1}\right)$$

La funzione energia della rete di Hopfield è:

$$E = -\frac{1}{2}\sum_{XY}\sum_{ij} T_{Xi,Yj}V_{Xi}V_{Yj} - \sum_{Xi} I_{Xi}V_{Xi}$$

Network Weights

The coefficients of the quadratic terms in the cost function define the weights of the connections in the network

$$T_{Xi,Yj} = -A\delta_{XY}(1-\delta_{ij})$$ {Inhibitory connection in each row}

$$-B\delta_{ij}(1-\delta_{XY})$$ {Inhibitory connection in each column}

$$-C$$ {Global inhibition}

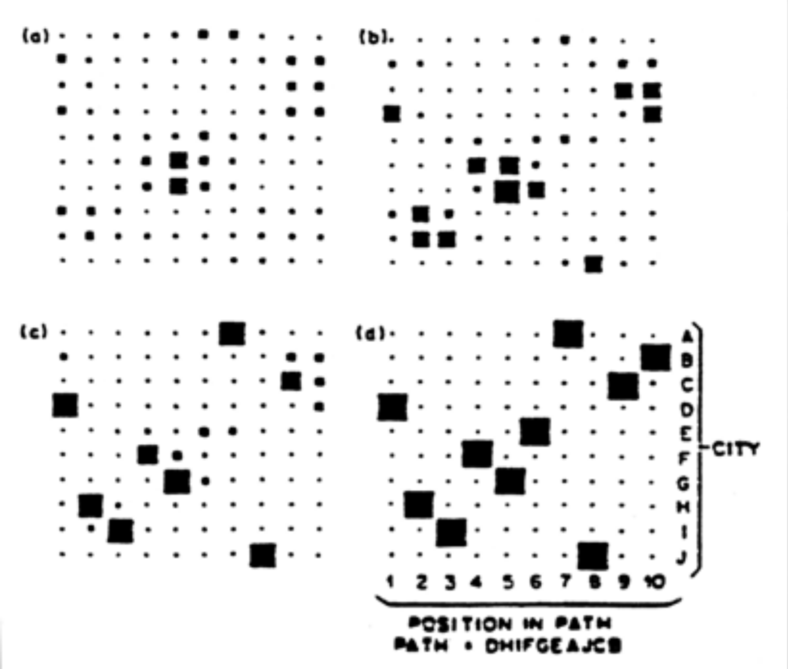
$$-Dd_{XY}(\delta_{j,i+1}+\delta_{j,i-1})$$ {Data term}

$$\delta_{ij} = \begin{cases} 1 & if \quad i = j \\ 0 & if \quad i \neq j \end{cases}$$

$$I_{Xi} = C_n$$ {External current}

Experiments

- 10-city problem, 100 neurons
- Locations of the 10 cities are chosen randomly with uniform p.d.f. in unit square
- Parameters: A = B = 500, C = 200, D = 500



- The size of the squares correspond to the value of the voltage output at the corresponding neurons.
- Path: D-H-I-F-G-E-A-J-C-B

TSP – A Second Formulation

Another way of formulating the TSP constraints (i.e., permutation matrix) is the following

$$E_2 = \frac{A}{2} \sum_X \left(\sum_i V_{Xi} - 1 \right)^2$$ row constraint

$$E_3 = \frac{B}{2} \sum_i \left(\sum_X V_{Xi} - 1 \right)^2$$ column constraint

The energy function becomes :

$$E = \frac{D}{2} \sum_X \sum_{Y \neq X} \sum_i d_{XY} V_{Xi} \left(V_{Yi+1} + V_{Yi-1} \right) + E_2 + E_3$$

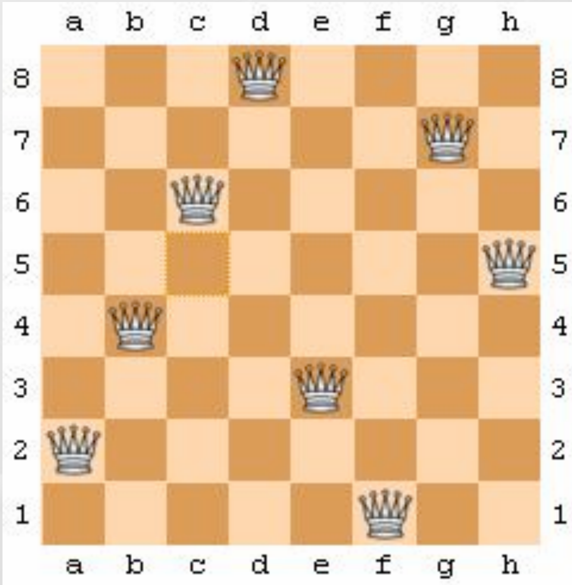
Advantage : less parameters (A,B,D)

The N-queen Problem

Build an *n* x *n* network whose neuron (*i*, *j*) is active if and only if a queen occupies position (*i*, *j*)

There are 4 constraints :

1. Only one queen on each row
2. Only one queen on each column
3. Only one queen on each diagonal
4. Exactly *n* queens on the chessboard



Network Weights

Following Hopfield's idea for the TSP, the weights become:

$$
\begin{aligned}
-T_{ij,kl} = {} & A\left(1-\delta_{jl}\right)\delta_{ik} + \\
& B\,\delta_{jl}\left(1-\delta_{ik}\right) + \\
& C + \\
& D\left(\delta_{i+j,k+l} + \delta_{i-j,k-l}\right)\left(1-\delta_{ik}\right)
\end{aligned}
$$

A ≡ inhibitory connection on each row

B ≡ inhibitory connection on each column

C ≡ "global" inhibitory connection

D ≡ inhibitory connection on each diagonal

Hopfield's Networks for Optimization

Shortcomings of the original formulation :

1) number of connection is $O(n^4)$ and number of units $O(n^2)$

2) not clear how to determine the parameters A, B, C, D

3) no theoretically guarantee that the solutons obtained are indeed "permutation matrices"

4) not clear how to avoid local minima

5) the relation between the original (discrete) problem and the continuous one holds only in one direction (that is, although each "discrete" solution corresponds to a solution in the continuous space, the converse needs not be true)

The Maximum Clique Problem (MCP)

You are given:

- An undirected graph $G = (V,E)$, where
  - $V = \{1,\ldots.,n\}$
  - $E \subseteq V \times V$

and are asked to

- Find the largest complete subgraph (clique) of G

The problem is known to be NP-hard, and so is problem of determining just the size of the maximum clique. Pardalos and Xue (1994) provide a review of the MCP with 260 references.

Some Notation

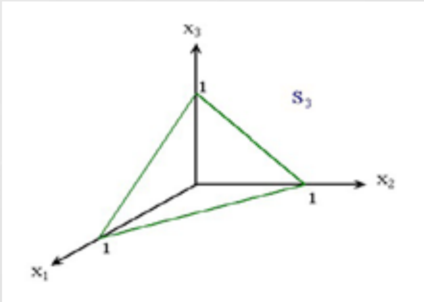
Given an arbitrary graph G = (V,E) with n nodes:

- If $C \subseteq V$, x^c will denote its characteristic vector which is defined as

$$x_i^c = \begin{cases} 1/|C|, & \text{if } i \in C \\ 0, & \text{otherwise} \end{cases}$$

- S_n is the standard simplex in R^n :

$$S_n = \left\{ x \in R^n : \sum_{i=1}^{n} x_i = 1 \text{ and } x_i \geq 0, \forall i \right\}$$



- A=(a_{ij}) is the adjacency matrix of G:

$$a_{ij} = \begin{cases} 1, & \text{if } v_i \sim v_j \\ 0, & \text{otherwise} \end{cases}$$

The Lagrangian of a graph

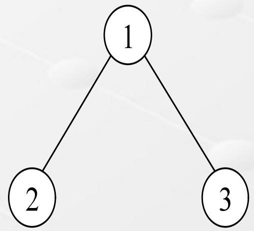
Consider the following "Lagrangian" of graph *G*:

$$f(x) = x'Ax = \sum_{i=1}^{n} \sum_{j=1}^{n} a_{ij}\, x_i\, x_j$$

Where a prime (') denotes transposition and *A* is the adjacency matrix of G.

Example:

$$f(\bar{x}) = \sum_{i,j \in E} x_i\, x_j$$



$$f(x_1, x_2, x_3) = x_1 x_2 + x_1 x_3$$

The Motzkin-Straus Theorem

In the mid-1960s, Motzkin and Straus (1965) established a remarkable connection between the maximum clique problem and the following standard quadratic program:

$$\begin{aligned} &\text{maximize} \quad f(\mathbf{x}) = \mathbf{x}' A_G \mathbf{x} \\ &\text{subject to} \quad \mathbf{x} \in \Delta \subset \mathbb{R}^n, \end{aligned} \tag{4.1}$$

where n is the order of G. Specifically, if $\mathbf{x}^*$ is a global solution of equation 4.1, they proved that the clique number of G is related to $f(\mathbf{x}^*)$ by the following formula:

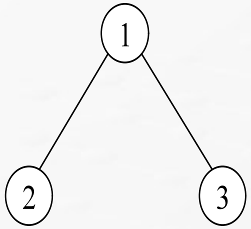
$$\omega(G) = \frac{1}{1 - f(\mathbf{x}^*)}. \tag{4.2}$$

Additionally, they showed that a subset of vertices C is a maximum clique of G if and only if its characteristic vector $\mathbf{x}^C$, which is the vector of Δ defined as

$$x_i^C = \begin{cases} 1/|C|, & \text{if } i \in C \\ 0, & \text{otherwise,} \end{cases}$$

is a global maximizer of f on Δ.[4] Gibbons, Hearn, Pardalos, and Ramana (1997), and Pelillo and Jagota (1995), extended the Motzkin-Straus theorem by providing a characterization of maximal cliques in terms of local maximizers of f on Δ.

Spurious Solutions

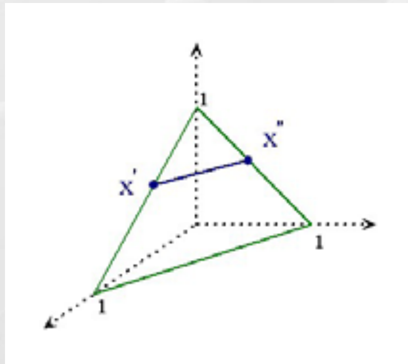


$$f(x_1, x_2, x_3) = x_1 x_2 + x_1 x_3$$

$$x' = \left(\frac{1}{2}, \frac{1}{2}, 0\right)^T \qquad x'' = \left(\frac{1}{2}, 0, \frac{1}{2}\right)^T$$

All points on the segment joining x' and x''

are "spurious" solutions.



Solution: add ½ over the main diagonal of A

$$A' = A + \frac{1}{2} I \quad \Longrightarrow \quad f(x) = x^T A' x \quad \Longrightarrow \quad f(x) = x^T \left(A + \frac{1}{2} I\right) x$$

The regularized Motzkin-Straus Theorem

THEOREM (Bomze, 1997)

Given $C \subseteq V$ with characteristic vector x^c we have:

- *C is a* **maximum clique** *of G IFF* x^c is a **global maximizer** of f in S_n

- *C is a* **maximal clique** *of G IFF* x^c is a **local maximizer** of f in S_n

- all local/global maximizers are strict and have the form of a characteristic vector of some subset of vertices (that is, no spurious solutions)

Evolutionary Games

Developed in evolutionary game theory to model the evolution of behavior in animal conflicts.

Assumptions

- A large population of individuals belonging to the same species which compete for a particular limited resource
- This kind of conflict is modeled as a game, the players being pairs of randomly selected population members
- Players do not behave “rationally” but act according to a pre-programmed behavioral pattern, or *pure strategy*
- Reproduction is assumed to be asexual
- Utility is measured in terms of Darwinian fitness, or reproductive success

Notations

- $J = \{1, \cdots, n\}$ is the set of pure strategies
- $x_i(t)$ is the proportion of population members playing strategy i at time t
- The state of population at a given instant is the vector $x = (x_1, \cdots, x_n)'$
- Given a population state x , the *support* of x, denoted $\sigma(x)$, is defined as the set of positive components of x, i.e.,

$$\sigma(x) = \{i \in J \ : \ x_i > 0\}$$

Payoffs

Let $A = (a_{ij})$ be the $n \times n$ *payoff* (or fitness) matrix.

a_{ij} represents the payoff of an individual playing strategy i against an opponent playing strategy j $(i, j \in J)$.

If the population is in state x, the expected payoff earnt by an i – strategist is:

$$\pi_i(x) = \sum_{j=1}^{n} a_{ij} x_j = (Ax)_i$$

while the mean payoff over the entire population is:

$$\pi(x) = \sum_{i=1}^{n} x_i \pi_i(x) = x'Ax$$

Replicator Equations

Developed in evolutionary game theory to model the evolution of behavior in animal conflicts (Hofbauer & Sigmund, 1998; Weibull, 1995).

Let $W = \left(w_{ij}\right)$ be a non-negative real-valued $n \times n$ matrix, and let

$$\pi_i\left(t\right) = \sum_{j=1}^{n} w_{ij}\, x_j\left(t\right)$$

Continuous-time version:

$$\frac{d}{dt} x_i\left(t\right) = x_i\left(t\right)\left(\pi_i\left(t\right) - \sum_{j=1}^{n} x_j\left(t\right)\pi_j\left(t\right)\right)$$

Discrete-time version:

$$x_i\left(t+1\right) = \frac{x_i\left(t\right)\pi_i\left(t\right)}{\sum_{j=1}^{n} x_j\left(t\right)\pi_j\left(t\right)}$$

Replicator Equations & Fundamental Theorem of Selection

S_n is invariant under both dynamics, and they have the same stationary points.

Theorem: *If* $W = W'$*, then the function*

$$F(x) = x' W x$$

is strictly increasing along any non-constant trajectory of both continuous-time and discrete-time replicator dynamics

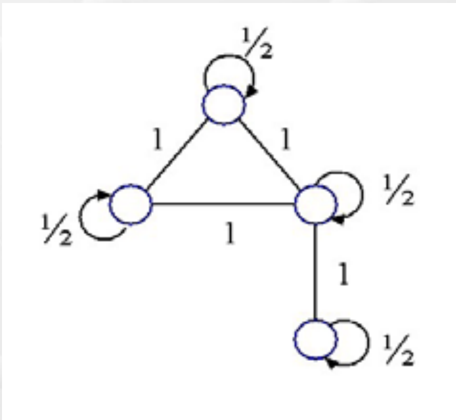
Mapping MCP's onto Relaxation Nets

To (approximately) solve a MCP by relaxation, simply construct a net having n units, and a $\{0,1\}$-weight matrix given by

$$W = A + \frac{1}{2} I_n$$

where A is the adjacency matrix of *G.*

Example:



The system starting from x(0) will maximize the Motzkin-Straus function and will converge to a fixed point x^* which corresponds to a (local) maximum of the Lagrangian.

Experimental Setup

Experiments were conducted over random graphs having:

- size: n = 10, 25, 50, 75, 100
- density: δ = 0.10, 0.25, 0.50, 0.75, 0.90

Comparison with Bron-Kerbosch (BK) clique-finding algorithm (1974).

For each pair (n , δ) 100 graphs generated randomly with size n and density ≈ δ.

The case n = 100 and δ = 0.90 was excluded due to the high cost of BK algorithm.

Total number of graphs = 2400.

δ \ n	10	25	50	75	100
0.10	0.99 (54)	0.99 (36)	0.99 (53)	0.97 (59)	0.92 (82)
0.25	0.99 (54)	0.99 (64)	0.99 (84)	1.00 (98)	0.97 (112)
0.50	1.00 (56)	0.99 (118)	0.99 (153)	0.96 (160)	0.90 (187)
0.75	1.00 (99)	1.00 (175)	1.00 (268)	1.00 (284)	1.00 (369)
0.90	1.00 (119)	1.00 (224)	1.00 (367)	0.99 (513)	----

$$Q = \frac{f_{ave} - f_{RE}}{f_{ave} - \alpha}$$

Values of Q-measure for various sizes and densities

Graph Isomorphism

Definition 1. *The association graph derived from graphs* $G' = (V', E')$ *and* $G'' = (V'', E'')$ *is the undirected graph* $G = (V, E)$ *defined as follows:*

$$V = V' \times V''$$

and

$$E = \left\{((i,h),(j,k)) \in V \times V \colon i \neq j,\ h \neq k,\ \text{and } (i,j) \in E' \Leftrightarrow (h,k) \in E''\right\}.$$

Theorem 1. *Let* $G' = (V', E')$ *and* $G'' = (V'', E'')$ *be two graphs of order* n*, and let* G *be the corresponding association graph. Then* G' *and* G'' *are isomorphic if and only if* $\omega(G) = n$*. In this case, any maximum clique of* G *induces an isomorphism between* G' *and* G''*, and vice versa. In general, maximal and maximum cliques in* G *are in one-to-one correspondence with maximal and maximum common subgraph isomorphisms between* G' *and* G''*, respectively.*

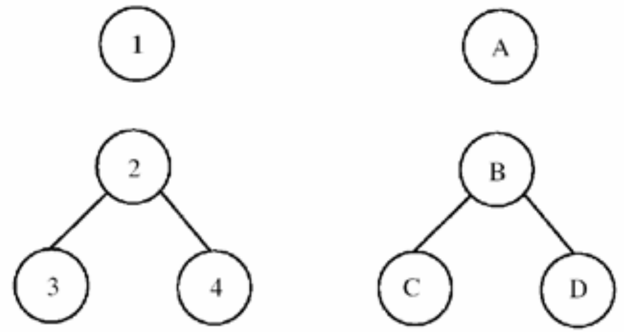


Figure 1: A pair of isomorphic graphs.

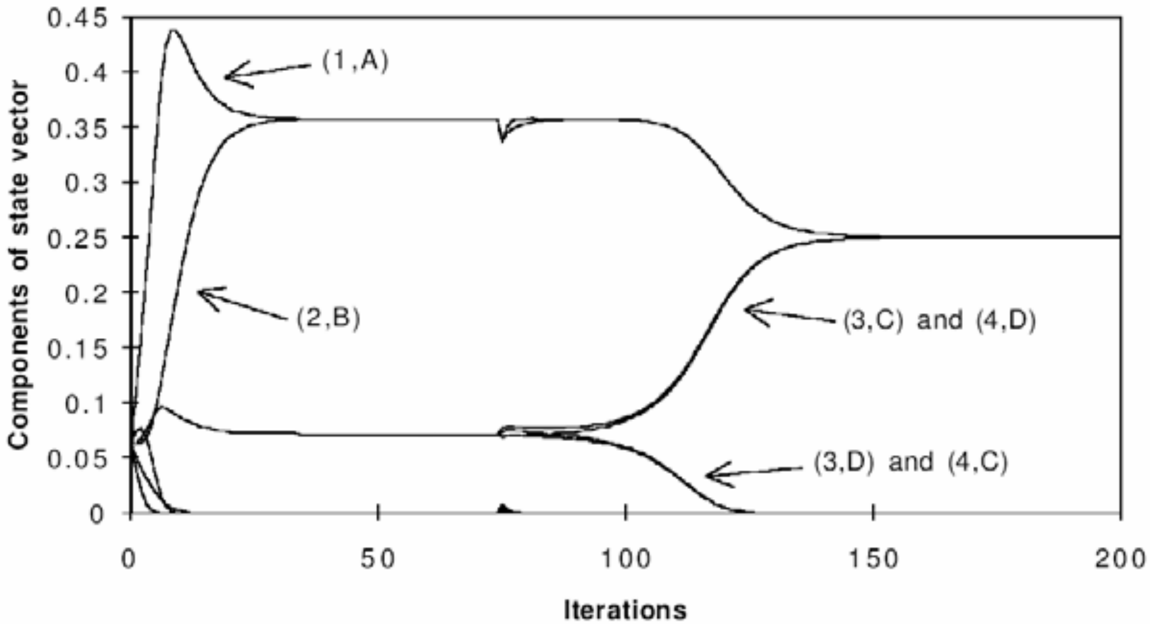
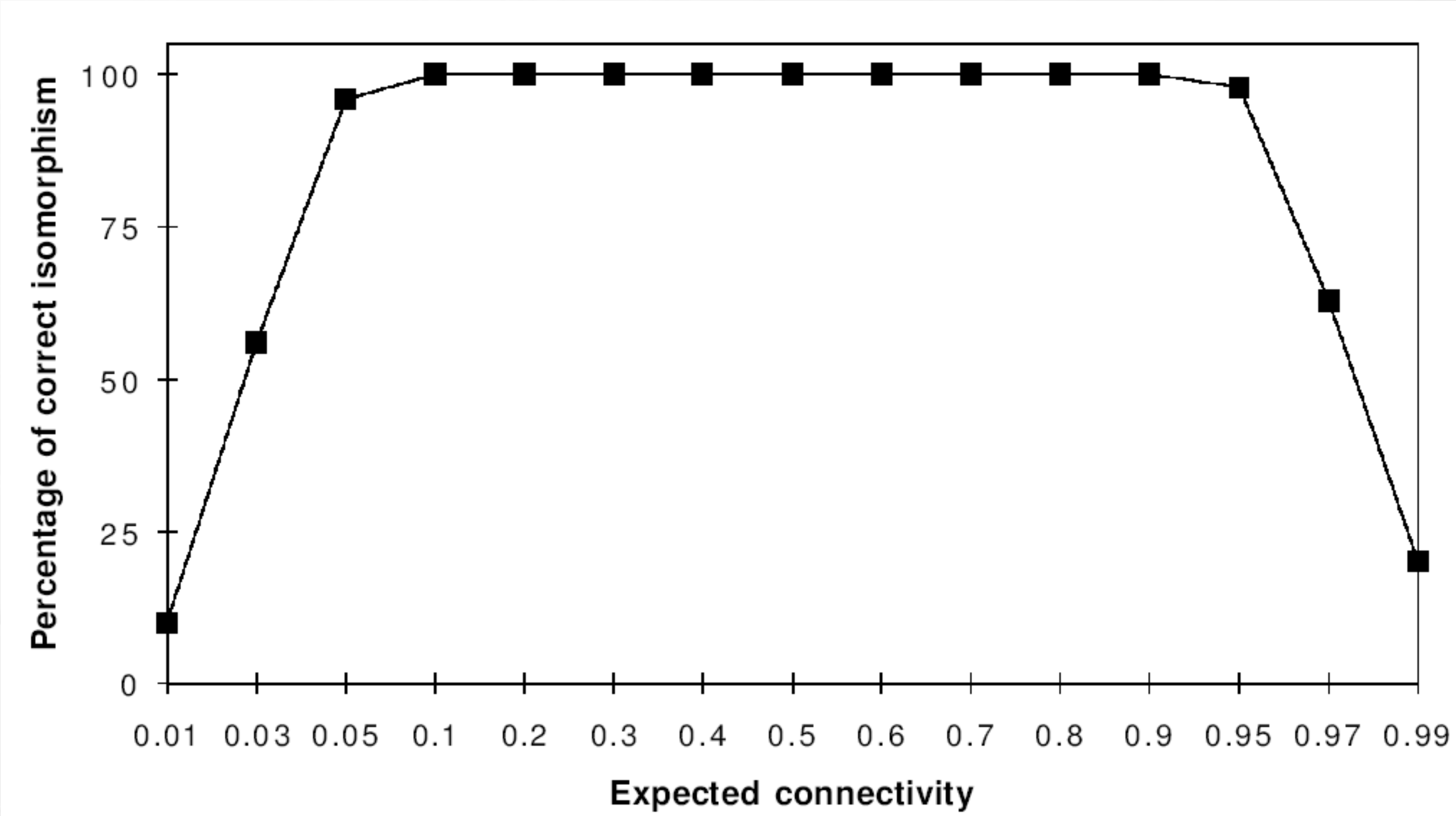
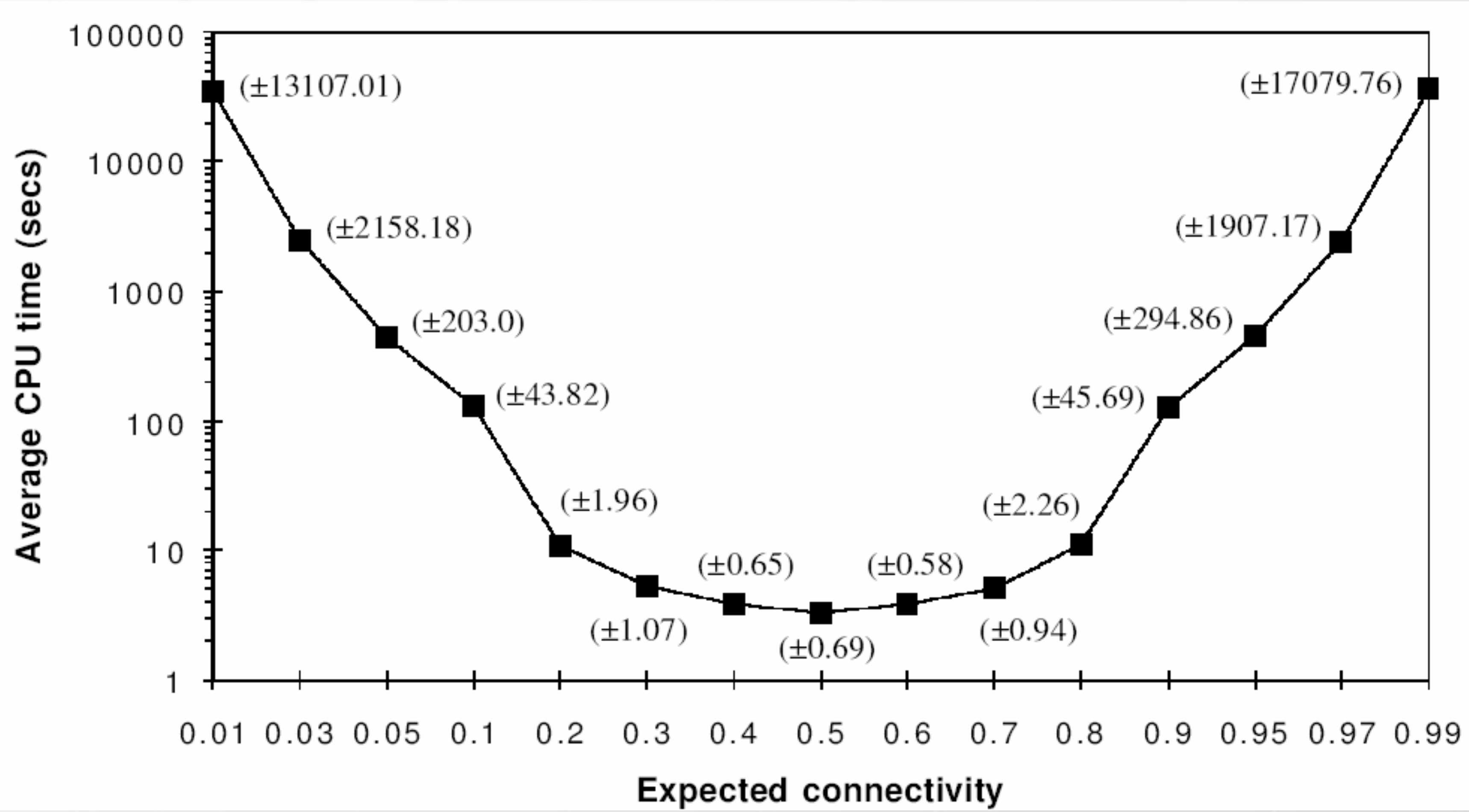


Figure 2: Evolution of the components of the state vector $\mathbf{x}(t)$ for the graphs in Figure 1, using the replicator dynamics (see equation 3.2).

Results


Percentage of correct isomorphism
100
75
50
25
0
0.01
0.03
0.05
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
0.95
0.97
0.99
Expected connectivity



100000
10000
1000
100
10
1
Average CPU time (secs)
(±13107.01)
(±2158.18)
(±203.0)
(±43.82)
(±1.96)
(±1.07)
(±0.65)
(±0.69)
(±0.58)
(±0.94)
(±2.26)
(±45.69)
(±294.86)
(±1907.17)
(±17079.76)
0.01 0.03 0.05 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 0.95 0.97 0.99
Expected connectivity