OPTIMAL RADAR PULSE SCHEDULING USING A NEURAL NETWORK

Alberto Izquierdo-Fuente (*), J.R. Casar-Corredera (**)

(*) ETSI. Telecomunicación - Universidad de Valladolid
C/ Real de Burgos s/n 47011 VALLADOLID SPAIN
Tf: +34 83 423260

(**) Dpto. Señales, Sistemas y Radiocomunicaciones. ETSI Telecomunicación-UPM
Ciudad Universitaria 28040 MADRID SPAIN
Tf: +34 1 5495700 (Ext. 206)

ABSTRACT

In a multifunction radar, the maximum number of targets which can be managed (maintain their tracks) is one of the main performance figures. The interleaving algorithms exploit the dead times between the transmitted and the received pulses to allocate new tracking tasks (transmit and receive pulses) thus increasing the capacity of the system.

Generally speaking, the interleaving of N targets is a complex problem (NP-complete type) and sub-optimal solutions are usually employed to satisfy the real-time constraints of the radar system.

In this paper we design an optimal interleaving algorithm based on a Hopfield network. We describe the general problem, define the network and select the criteria to design the weights. Finally as an example, we simulate a simple scenario with five targets.

INTRODUCTION

A multifunction radar, which is based on an electronically phased array, can steer the antenna beam to any direction almost instantaneously. Because of that, a multifunction radar can carry out simultaneously two kind of tasks: Surveillance and Tracking.

Once the surveillance tasks have been defined, the objective is to maximize the number of targets to track on the available time (total time minus surveillance time). Since we have an estimate of the range (distance) of the targets we know the dead time between the transmitted pulse and the received (reflected) pulse. Therefore we can interleave other pulses in these intervals.

The problem of interleaving the transmitted and received pulses of N targets is NP-complete. We have N! possibilities to order the pulses, and sometimes only one is feasible. The computational burden increases with the number of targets and usually sub-optimal solutions are employed.

These solutions sort the targets with a fixed criterion like the Farina & Studer algorithm [1], which orders the targets by decreasing range and try to allocate each received pulse next on the right of previous one. Other solutions like that of the Correlation algorithm [2] find the first available time slot in the temporal frame to allocate the next task.

In Fig. 1 we present a typical example of an interleaving algorithm, where pri represents the interval between consecutive transmitted pulses.

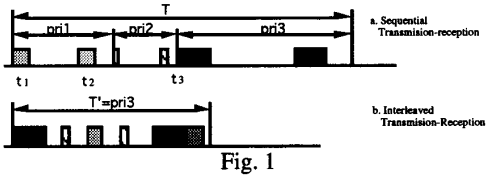


Fig. 1

We have three tasks to interleave. If the tasks are not interleaved (sequential scheduling) they spend a total time T= pri1 + pri2 + pri3. On the other hand, we can interleave tasks 1 and 2 in the dead time of task 3. Note the reduction in the necessary time from T to T'.

This paper handles the interleaving problem from a neural network viewpoint. For a given time interval, usually the pri of the farthest target, we want to find an algorithm which maximises the number of allocated pulses probing on all possible ordinations of the targets.The solution given by our algorithm should be the transmission order and the delay for each target in relation to a common time reference.

INTERLEAVING BASED ON NEURAL NETWORKS

The problem can be formulated in the following terms:

Input information

N tasks to allocate, having each one two τ_i μs wide pulses (transmit & receive), separated T_i sec.(gap)

0-7803-1901-X/94 $4.00 ©1994 IEEE

Constraints

a) Neither of the pulses can be overlapped.
b) A task can be delayed T_r seconds provided the corresponding received pulse can be also scheduled. in the available time interval.

Output information

Order and relative delay for each task allocated.

To pre-process this information, we can simplify the input information, assuming:

a) The time axis is discrete, with Ts sec. slots.
b) The pulses have a constant width τ=Ts (time slot)
c) Gap between pulses T_i has to be an integer number of times the basic time slot: T_i=k*Ts
d) Delay for each task has to be an integer number of times the basic time slot: T_r=j*Ts
e) Available time interval has to be an integer number of times the basic time slot: T_t=i*Ts

Now we can figure out a paradigm as that shown in Fig.2. Where the neural network is built with L levels, each level having N neurons. Each level represents a time slot and the neurons at that level the tasks candidates to be scheduled in that slot. In this way we will have a neural network with N*L elements where neuron (i,j) corresponds to time slot j and task i.

The resulting paradigm is a dynamic Hopfield network where the weights are designed with the input information (gaps) and the constraints (overlap and maximum delay), and the final state of the neurons contain the output information.

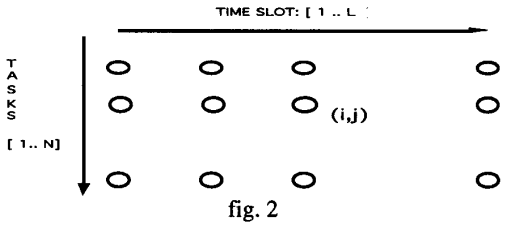


fig. 2

We only represent the transmitted pulse position since the received pulse position is the transmitted position plus the gap. After convergence, there will be as many neurons activated as tasks to allocate. For each row of neurons there will be only one activated neuron, which will indicate the scheduled position of the transmitted pulse.

Design of Weights

1) The $2L^2$-L weights that connect every pair of the neurons will be either null or negative,

2) Let T1 and T2 be two tasks. Let us assume that the transmitted pulse of T1 is allocated in the time slot represented by neuron Ni (i-th slot) and that the transmitted pulse of T2 is allocated in the time slot represented by neuron Nj (j-th slot). We will assign a null to the weight which links Ni and Nj if no one of the transmitted and received pulses overlap. Otherwise, we will assign a negative value to the weight.

3) For a task, assuming that the transmitted pulse is allocated in the position represented by the neuron N_i, we assign negative values to all the weights of that neuron, if the received pulse position exceeds the available time.

4) On the other hand, as only one neuron can be activated per task, we will assign negative weights between the neurons of a same task.

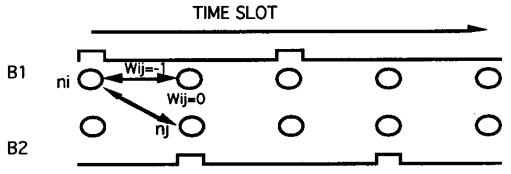


fig. 3

Dynamic node equations

Neurons are activated asynchronously with a random pattern as follows:

$$u_{(i,j)} = f\left(\sum_{k=1}^{k=N}\sum_{l=1}^{M} w_{(i,j)} u_{(i,j)} + v_{(i,j)}\right)$$

where:

u(i,j) neuron output state (i,j)
v(i,j) neuron input excitation (i,j)
w(i,j) weights from neuron (i,j) to all neurons

We assume that all neurons are activated in an initial input state. All arrangement of tasks are allowed.

$$U = \begin{bmatrix} 1 & \cdots & 1 \\ \vdots & & \vdots \\ 1 & \cdots & 1 \end{bmatrix}$$

The input excitation is equal for all neurons and decreases gradually after N iterations.

$$if \quad k = nN \quad v_{(i,j)} = v_{(i,j)} - \Delta v \quad n = [1,2,...]$$

In this way, we manage to cold the network until we obtain one of the possible solutions. Note that for v(i,j)=0, only those neurons which satisfy the constraints can be activated.

Finally the function f(), is a hard-limiter:

$$f(u) = \begin{cases} 0 & u < 0 \\ 1 & u \geq 0 \end{cases}$$

EXAMPLE

As an example we have designed a network for N=5 tasks and a time axis with L=14 slots. Thus the total number of neurons will be N*L = 60 neurons.
The targets (tasks) are sorted in range decreasing order, with gaps from one to five time slots, as we can see in figure 4 (where T represents the transmitted pulse and R the received pulse).

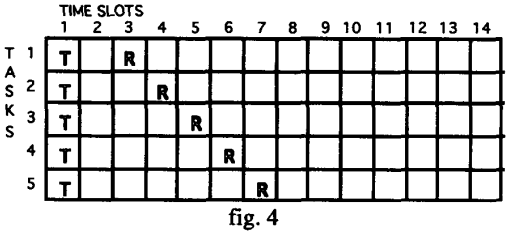


fig. 4

The initial input state is given by:

$$U = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

For setting the weights we used the values:

$$W = \begin{cases} 0 \\ -1.5 \end{cases}$$

We have performed, in a random way, 10*5*14 iterations, beginning with an input excitation v=1,5 and decreasing it in steps of 0.1 every 60 iterations.

In Fig. 5 we show the evolution versus time of the output state for a specific simulation experiment.

v = 1.4

$$U = \begin{bmatrix} 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

v = 1.3

$$U = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix}$$

v = 1.2

$$U = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

v = 1.1

$$U = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

v = 1.0

$$U = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

v = 0.9

$$U = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

v = 0.8

$$U = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

v = 0.5

$$U = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Fig.5

The final solution turns out to be that shown in Fig. 6

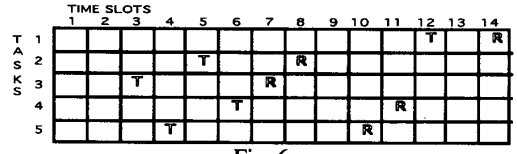
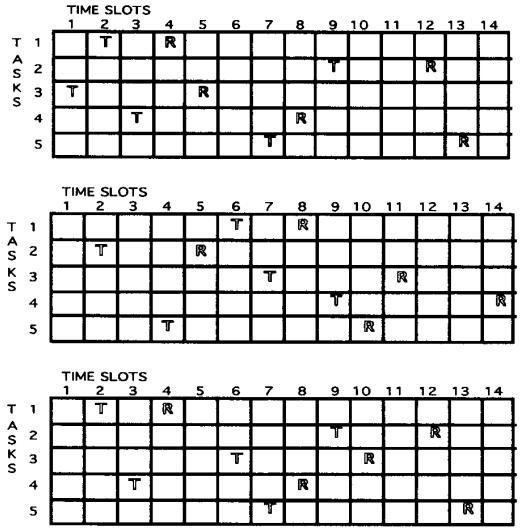


Fig. 6

Finally we present a final state set for different random patterns.



CONCLUSIONS

We have presented a new approach to interleaving based on a Hopfield's neural network. Under some simplifications we have designed the network layout and we have established a criterion for designing weights and neuron activations (colding process . For a simple example and a set of random patterns the results have been shown to be excellent. More complex scenarios have been tested with similar results.

The next step will be to evaluate the computational burden and efficiency in relation to classical algorithms.

REFERENCES

[1] A. Farina and P. Neri. " Multitarget interleaved tracking for phased-array radar " IEE Proc. Vol 127 Pt. F, No 4 Aug. 1980

[2] Alberto Izquierdo y J.R. Casar Corredera. " Análisis y selección de algoritmos de entrelazado de pulsos para radares multifunción". VII Simposium Nacional de la Unión Cientffica de Radio.URSI-92. Málaga (Spain). Sept, 1992. pp. 934-938.

[3] Richard P. Lippmann. " An introduction to computing with neural nets" IEEE ASSP Magazine April 1987

[4] Don R. Hush and Bill Horne. " An overview of neural networks: Part I / Part II" Revista Informática y Automática. Vol 25, Num. 1-2. March-June 1992.