Hopfield Networks, Neural Data Structures and the Nine Flies Problem:
Neural Network Programming Projects for Undergraduates

John G. Keating

Department of Computer Science,
St. Patrick's College, Maynooth, Co. Kildare, IRELAND.

JNKEATING@VAX1.MAY.IE

ABSTRACT

This paper describes two neural network programming projects suitable for undergraduate students who have already completed introductory courses in Programming and Data Structures. It briefly outlines the structure and operation of Hopfield Networks from a data structure stand-point and demonstrates how these type of neural networks may be used to solve interesting problems like Perelman's Nine Flies Problem. Although the Hopfield model is well defined mathematically, students do not have to be very familiar with the mathematics of the model in order to use it to solve problems. Students are actively encouraged to design modifications to their implementations in order to obtain faster or more accurate solutions. Additionally, students are also expected to compare the neural network's performance with traditional approaches, in order that they may appreciate the subtleties of both approaches. Sample results are provided from projects which have been completed during the last three-year period.

INTRODUCTION

The *N-Queens Problem* (NQP) is concerned with placing N queens on a square N×N chessboard in such a way that no more than one queen appears in any row, column or diagonal of the board. Despite the fact that only one queen appears in any row or column, finding one (or all) solutions to the problem is a reasonably time-consuming task as there are still N! arrangements along the diagonals to be examined. The NQP is just one of many combinatorial optimization problems which have resisted centuries of attempts at providing analytical solutions; in the case of the NQP, this requires finding some systematic method of placing a queen on the board in order to preclude attacks by future placements. Traditionally, this problem may be solved using Backtracking (see McCracken and Salmon, 1987), but in recent times, methods of obtaining solutions using Neural Networks have appeared in the literature (Shackleford, 1989; Dory, 1990; Mandziuk and Macukow, 1992).

An interesting variation of the NQP has been presented by Perelman in his wonderful book "Fun with Maths and Physics". It is the so-called *Nine Flies Problem* (NFP), and may also be solved by computer using Backtracking or with a Hopfield network. The problem is as follows: "Nine Flies are sitting on a chequered window curtain (see Figure 1). They happened to have arranged themselves so that no two flies are in the same row, column or diagonal. After a while three flies shifted into neighbouring, unoccupied cells and the other six stayed in the same place. Curiously enough, the nine flies still continued to be arranged so that not a single fly appeared in the same direct or oblique line. Find which three flies moved, and the new arrangement" (after Perelman, 1987). The proposed projects require students to solve the NQP and the NFP using using both Hopfield Networks and Backtracking.

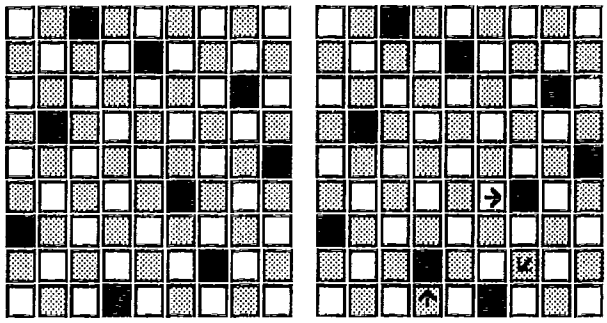

Figure 1. Perelman's Nine Flies Problem. The diagram on the left shows the arrangement of the flies initially (the squares occupied by flies are shown as black squares). The diagram on the right shows which three flies moved, and the new arrangement.

Schweller and Plagman (1989) and Wallace and Wallace (1991) agree that it is extremely important that students are introduced to both interesting and active research areas early in the curriculum, and strongly emphasise the importance of exposing them to neural network programming. It also familiarizes students with a non-traditional programming paradigm. They have found that successful completion of such assignments gives students

a sense of accomplishment, demystifies neural networks and encourages active student participation in meaningful research and investigation. I believe that students will gain a full appreciation of problem-solving using neural networks, only if they solve first problems using traditional methodologies and then compare the methods used with the connectionist approach.

Junior students who have completed introductory courses in Programming and Data Structures should, having had two lectures on associated background information, complete both projects in about 8-10 hours. The projects may also be assigned to senior students taking a course in Neural Networks. These students could be given a statement of the problem together with a number of introductory and more detailed references from which they may determine, for themselves, how both problems might be solved.

HOPFIELD NETWORKS - BACKGROUND

The Hopfield network (Hopfield and Tank, 1985, 1986; Tank and Hopfield, 1987) is usually implemented as a single layer of laterally connected binary-valued or analog neurons. All neurons in the network function in the exact same manner: each neuron sums it's weighted input signals and produces an output based on it's transfer function. Discrete neurons compare the weighted sum of inputs against some threshold and output a 1 if the weighted sum is greater than the threshold, otherwise a 0 is output. Continuous neurons, on the other hand, produce an output (usually between 0 and 1) based on the weighted sum and a transfer function which is usually a nonlinear symmetric sigmoid.

Hopfield networks have shown to be useful both as both autoassociators and as optimizers. In order to create an autoassociater, each neuron is connected to all other neurons in the network, and is trained by directly calculating the network weights, or connection matrix, using a qualitative application of the Hebb rule (Hebb, 1949). It is necessary, therefore, that all of the patterns to be learned by the network must be available at the very outset, and these patterns form the stable states of the network. In general, the maximum number of patterns which may be stored by a Hopfield network is approximately 0.15 times the number of neurons in the network. When used as an optimizer, the network structure and weights usually depend on specific constraints associated with the problem under consideration. Stable states of the network correspond to solutions of the problem, and these states may be found by following a path of energy minimization for the total energy associated with the network.The rest of this section briefly summarises Hopfield's treatment of laterally-connected neural networks.

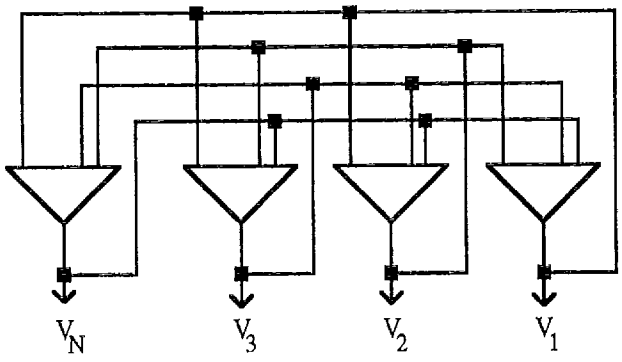

Figure 2. A laterally-connected Neural Network. Note that the input to each neuron is the output of all other neurons in the network. These inputs are multiplied by the connection weights (T_{ij}) which may be excitatory or inhibitory (after Dory, 1990).

Consider a laterally-connected network consisting of N neurons as shown in Figure 2, and let T_{ij} represent the connection weight between neuron i and neuron j. T_{ij} may be either positive (an excitatory stimulus) or negative (inhibitory stimulus). The sum of the inputs (or the input potential) to any neuron i is given by

$$U_i = \sum_{j=1}^{N} T_{ij} V_i, \; j \neq i \qquad (1)$$

where V_j is the output (or the output potential) of the other N-1 neurons in the network. In the case of continuous Hopfield neurons, the output potential is usually given by

$$V_i = \frac{1}{(1 + e^{-GU_i})} \qquad (2)$$

where G is the gain of the sigmoid function. For sufficiently large values of G, the neuron is of binary character as $V(U)$ is approximately represented by

$$V(U) = \begin{cases} 0 \text{ if } U<0 \\ 1 \text{ if } U>0 \end{cases}$$

Hopfield noticed a simularity between the network's behaviour and certain physical systems (Spin Glasses), and applied some results from statistical physics in order to examine the progression of the network towards stability. He associated an energy, E, with the state of the network, and showed that as the network approaches stability, the energy function decreases towards a local minimum. For the network shown in Figure 2, the energy function is given by

$$E = -\frac{1}{2} \sum_{j=1}^{N} \sum_{i=1}^{N} T_{ij} V_i V_j \qquad (3)$$

This approach generated renewed interest in neural networks and has, without doubt, been responsible for the advancement of the field.

Training a Hopfield network to be used as an optimizer is a two stage process: (i) choose a network structure which

appropriately models the problem, and (ii) find the weights which minimize the network's energy function. The latter task is usually the harder, as it necessitates finding an expression for the energy of the system, and then minimizing it. This is usually achieved by initializing the weights to some random value and repetedly adjusting them until the energy is minimized. Weight adjustment should be performed systematically using a method which ensures the network does not get bound in some local energy well, for example, *Simulated Annealing*.

Recently, this method was used effictively by Mandziuk and Macukow (1992) to find solutions to the NQP. They represented the NQP as a two-dimensional interconnected array of Hopfield neurons whose stable states corresponded to solutions. The energy associated with this network is given by

$$2E = AS_1 + BS_2 + CS_3 + n\_S_4 \qquad (4)$$

where A, B, C and $n\_$ are positive constants, and S_1, S_2, S_3 and S_4 are sums responsible for interactions between rows, columns and diagonals in the network. This method is similar to that used by Hopfield and Tank (1985) to obtain solutions to the *Travelling Salesman Problem*.

NEURAL DATA STRUCTURES

I encourage students to use the data-structure approach outlined by Shackleford (1989) to obtain solutions to the NQP and NFP rather than the mathematical method described above. This approach is readily understood by almost all students, and is especially useful to use with students who have difficulty understanding the selection of an appropriate energy equation and method of minimization. Shackleford describes a method whereby N neurons are combined to form a laterally connected layer (as shown in Figure 1) with all of the T_{ij} set to -1. This system of interconnections is known as *lateral inhibition*, a term borrowed from biologists. All neurons in the layer have the same gain, G, and recieve an additional excitatory stimulus K which tends to drive each neuron's output towards 1. These type of neural data structures are called *N-flops* as they have N stable states (one neuron outputting a 1 and all others outputting 0), and may be represented diagramaticaly as shown in Figure 3.

If the neuron outputs are initialized to some random starting value between 0 and 1, then with time, one neuron will start to predominate, thus forcing other neurons in the layer towards 0. This in turn lessens the inhibitory effect on the predominating neuron and allows it to use the energy derived from the K input to drive it towards 1. This one-of-N behaviour is dependent on the values of G, K and N, otherwise more than one neuron will predominate and the network will behave as an M-of-N behaviour may be determined from parameter sensitivity studies. Typical values of G and K for an 8-flop with

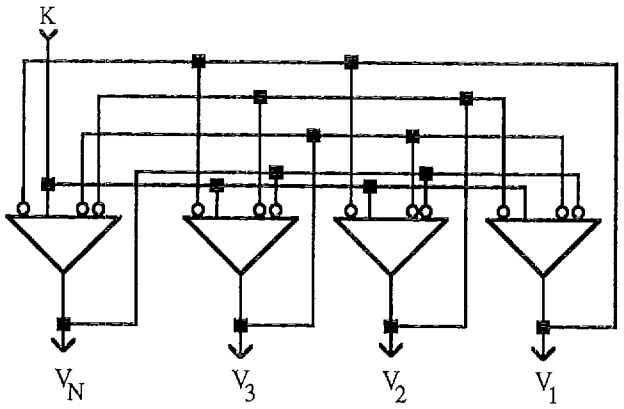

Figure 3. An N-flop (Shackleford, 1989). The layer is just a Hopfield network with connection weights equal to -1, in other words all lateral connections are inhibitory (depicted by circles on the inputs). Every neuron in the network also recives the same excitatory input, K.

lateral inhibition are 7.0 and 0.75 respectively (Shackleford, 1989).

N-flops may also have the additional consraint that the sum of all of the outputs will be close to some value; in the case of the N-flop this should be 1. This constraint is called *global inhibition*, and is implemented as follows: (i) compute the sum of all outputs, (ii) subtract the desired total value from this sum, and (iii) send the final value to an ihibitory input on all neurons in the N-flop. The net effect of global inhibition is to drive the outpts towards 1, while that of lateral inhibition is to ensure a one-of-N state. In fact, Shackleford (1989) has shown that global inhibition also increases the range of values of G and K for which one-of-N states may be obtained.

To solve the NQP using N-flops, take N^2 neurons and arrange them in N rows of N neurons, with each neuron having inhibitory connections to the other neurons in its row, column and diagonals. This data structure is usually referred to as an *N×N-flop*. Solutions to the NQP are obtained by initializing all the neurons to some unstable state, and repeatedly computing the next state of the network until stability is reached. In practice, however, the network may not be able to place all N queens on the board, as it becomes stuck in some local energy minimum. This usually means that the network does not have sufficient energy to return from the energy well. An N×N-flop may also be used to solve the NFP - it differs from the NQP in the way the data structure is used to obtain solutions.

THE N-QUEENS PROBLEM

Hopfield network solutions to the NQP should be readily found using the method outlined in the previous section provided the following preliminary studies are carried out first:

(i) Write and debug a program which implements an N-flop for variable N, G and K.

(ii) Ensure that the N-flop behaves as it should, ie. that each of the N states has an equal probability of occurring. A common mistake made by students is to update the N-flop sequentially, which results in one state occurring with a probability greater than $1/N$. The students should be reminded that each neuron should be computing an output continiously, and in parallel with the other neurons. Parallelism may be simulated by repeatly choosing a neuron at random and computing its output - iteration stops when stability is reached, ie. when the output of all neurons do not change with time.

(iii) Carry out a parameter sensitivity simulation of an 8-flop for both lateral inhibition and global inhibition cases, and confirm that this simulation produces similar results to those in the Shackleford paper for various combinations of G and K.

(iv) Before using an 8×8-flop to find solutions to the NQP, students should carry out a parameter sensitivity simulation for to determine suitable values for G and K. This is necessary because the 8×8-flop is actually 64 interdependent N-flops with N in the range 22 to 28. This investigation shows that it is reasonable to use the same G and K for every neuron in the network.

The network should be run a number of times (say 100) and the following items should be recorded: (i) the percentage number of solutions found, (ii) the number of different solutions found and (iii) the time taken to find a solution. The accumulation of these results can be combined with parameter sensitivity tests to produce network performance tables for both lateral and global inhibition cases. The tables may then be used to compare the method against performance tables obtained using a Backtracking method. Some of the simulation data for the Eight Queens Problem are given in Tables 1, 2 and 3.

G	$K = 0.5$	$K = 1.0$	$K = 1.5$	$K = 2.0$	$K = 2.5$
2.0	-	-	-	7	21
2.5	-	-	43	42	21
3.0	-	39	43	38	20
3.5	-	36	41	31	21
4.0	-	35	36	29	20
4.5	25	28	31	28	5
5.0	24	32	22	27	2
5.5	16	27	22	25	1
6.0	13	24	18	23	1
6.5	12	22	19	22	-
7.0	13	22	19	19	-
7.5	10	22	14	20	-
8.0	9	25	16	19	-
8.5	8	19	17	17	-
9.0	8	22	15	14	-
9.5	7	23	15	18	-
10.0	9	19	15	15	-

Table 1. Simulation of Eight Queens Problem using a Hopfield Network: *Lateral Inhibition* case. The table gives the number of solutions found in 100 trials for various combinations of the network parameters G and K. Dashes represent combinations of G and K for which no solutions were found.

G	$K = 0.5$	$K = 1.0$	$K = 1.5$	$K = 2.0$	$K = 2.5$
2.0	-	-	42	44	49
2.5	-	37	37	39	36
3.0	-	37	41	36	41
3.5	-	34	36	37	40
4.0	29	30	32	34	38
4.5	25	31	37	34	35
5.0	26	30	31	32	32
5.5	21	30	33	30	30
6.0	18	27	35	29	25
6.5	13	27	36	33	19
7.0	12	27	41	25	20
7.5	11	25	42	28	16
8.0	12	26	42	28	17
8.5	9	24	43	25	17
9.0	10	25	40	30	14
9.5	10	23	45	31	18
10.0	10	23	46	29	18

Table 2. Simulation of Eight Queens Problem using a Hopfield Network: *Global Inhibition* case. The table gives the number of solutions found in 100 trials for various combinations of the network parameters G and K. Dashes represent combinations of G and K for which no solutions were found.

Tables 1 and 2 show that both lateral and global inhibition are equally good at obtaining solutions: in the tests networks obtain solutions less than 50% of the time. The global inhibition simulation, however, performed better over a greater range of G and K. The performance of these networks are considerably worse than the one described by Mandziuk and Macukow (1992), but the approach is more easily understood and implemented by students unfamiliar with this programming paradigm.

Shackleford (1989) makes the point that having examined of the solutions to the NQP it becomes evident that queens are usually a Knight's Jump away from each other, and that if this was incorporated into the simulation it might improve the quality of the results. This task has been assigned as an additional exercise to students who have had little difficulty with the other tasks. Each neuron in the network gets an extra excitation from every other neuron a knight's jump away in addition to its usual inhibitory stimulus. Neurons on the border of the chessboard get an excitatory stimulus from wraparound neurons a knight's jump away on the other borders.

Table 3 shows a network performance table for a global inhibition simulation with an additional knight's jump excitation applied to every neuron. The maximum percentage number of solutions is 69% (K=2.5, G=2.5), which is a significant increase over the case where no excitation was used. It is evident from this simulation that the performance of the network may be improved considerably with simple but elegant changes.

G	$K = 0.5$	$K = 1.0$	$K = 1.5$	$K = 2.0$	$K = 2.5$
2.0	-	30	57	63	64
2.5	11	59	59	63	69
3.0	46	46	53	48	53
3.5	38	52	46	47	46
4.0	38	54	41	44	41
4.5	41	36	50	42	41
5.0	27	33	36	39	45
5.5	34	45	49	35	38
6.0	27	48	25	31	33
6.5	36	40	42	41	26
7.0	29	47	37	28	31
7.5	28	47	25	37	22
8.0	30	58	28	29	29
8.5	21	51	44	28	35
9.0	26	66	41	29	25
9.5	27	63	41	30	25
10.0	21	57	36	32	28

Table 3. Simulation of Eight Queens Problem using a Hopfield Network: *Global Inhibition* plus *Knight's Jump Excitation* case. The table gives the number of solutions found in 100 trials for various combinations of the network parameters G and K. Dashes represent combinations of G and K for which no solutions were found.

Having run the simulations, it should be obvious to the students that the critical difference between the Hopfield Network and Backtracking is concerned with finding all solutions to the NQP. The Hopfield network does not guarantee to deliver all solutions, whereas the Backtracking method does. This has also been found by Mandziuk and Macukow (1992) who used a more sophisticated method to obtain solutions than has been outlined here. As N increases, however, the Hopfield network tends to a single solution faster than the Backtracking method.

THE NINE FLIES PROBLEM

This problem may be modelled using a 9×9-flop where each neuron represents a square on the curtain. An output close to one corresponds to the presence of a fly on the square, and an output close to zero corresponds to the absence of a fly. A neural network solution to the problem may be found using the procedure shown in Figure 4. This method produced the required solution after four iterations of the procedure (about one minute using a C implementation running under UNIX in a 486-based workstation). Students should be able to make suitable modifications to their NQP programs, in order to find a solution to the NFP. As before, a parameter sensitivity test for a 9×9-flop should be caried out in order to find the most suitable values of G and K.

A Backtracking solution for the NFP was also developed, and a performance evaluation for both methods was carried out by the students. The results show that the Backtracking method outperforms the Hopfield network

```
1   Initialize neuron outputs with
    a known solution.

2   Randomly choose three neurons
    and fix their output at zero.

3   For each chosen neuron,
    initialize adjacent neurons
    with random output.

4   Iterate network until
    stability has been reached

5   If solution has not been
    found then goto 1
```

Figure 4. A Procedure which uses a Hopfield Network to find solutions to the Nine Flies Problem (NFP).

approach on a number of counts: (i) it provides all solutions to the NFP, (ii) it is faster than the Hopfield network, and (iii) the Hopfield network cannot determine if there is more than one solution. Using the Backtracking method it was possible to find all possible starting states for the NFP and all solutions for each of these starting states. It is not possible to achieve this rate of success using a Hopfield network.

CONCLUSION

I have described two neural network programming projects suitable for undergraduate students - they are required to solve the N-Queens Problem and Perelman's Nine Flies Problem using a Hopfield Network and compare the method's performance with a more traditional method. The students determine for themselves the benefits and limitations of neural network approaches to problem solving.

This project has proved to be very successful as the students feel that they are participating in meaningful and interesting research in the area of Neural network computing. They are encouraged (i) to familiarize themselves with various approaches to problem solving and (ii) investigate the benefits and limitations of each approach and (iii) make and informed choice based on their own research. I believe this research certainly highlights the importance of introducing students to new programming paradigms early in the curriculum, and giving them the opportunity to carry out research tasks as early as possible.

ACKNOWLEDGEMENTS

I would like to express my thanks to my students for their valuable comments and contributions over the past three years.

****Hopfield Networks Continued On Page 40****

```
procedure tshape.draw(view : surface);
  var
    x, y : real;
    x1, y1, x2, y2, x3, y3 : integer;
  begin
    x1 := trunc(xspot/view.rtn_xfactor);
    y1 := trunc(yspot/view.rtn_yfactor);
    x2:=x1+trunc(side3/view.rtn_xfactor);
    y2:=y1;
    x:=(sqr(side3)+sqr(side1)-sqr(side2))
        /(2*side3);
    y:=sqrt(sqr(side1) - sqr(x));
    x3:=x1+trunc(x/view.rtn_xfactor);
    y3:=y1-trunc(y/view.rtn_yfactor);

    line(x1,y1,x2,y2);
    line(x2,y2,x3,y3);
    line(x3,y3,x1,y1);
    readln;
  end;
```

Notice that (x3,y3) represents the 'top' point of the triangle so y is subtracted from y1 to go 'up' the screen.

Cshape, rshape, and sshape are all created in similar, and much easier, fashions. The main program asks for the horizontal screen dimension which is passed into a surface object using its method get_scale. Similarly, the dimensions of the particular figures are retrieved and stored using their inherited methods get_sides or get_radius in the case of a circle.

```
var
  screen : surface;
  tri : tshape;
  s1, s2, s3 : real;
  size : real;
  x, y : real;
  grdriver, grmode : integer;

begin
  clrscr;
  write('What is the horizontal ');
  write('dimension of the screen? ');
  readln(size);
  screen.get_scale(size);

  write('What are the dimensions ');
  write('of your triangle, largest');
  write(' side last? ');
  readln(s1, s2, s3);
  tri.get_sides(s1,s2,s3);

  write('Where (in & down) do you ');
  write('want the base located ');
  write('- left corner? ');
  readln(x,y);
  tri.position(x,y);

  detectgraph(grdriver,grmode);
  initgraph(grdriver,grmode);
  tri.draw(screen);
```

The above code shows a partial listing needed to draw a triangle. A fully implemented program would allow graphical representation of any of the four shapes using any real-world dimensions. When complete, this graphing problem represents a fine classroom example of object extensibility.

References

Borland International Inc. (1990) Turbo pascal version 6.0 user's guide. Scotts Valley, CA: Author.

Dixon C. (1991, June). An introduction to object-oriented programming through turbo pascal. SIGCSE Bulletin, pp. 33-35,38.

The Whitewater Group. (1988). Introduction to object-oriented programming. Evanston, Il: Author.

Hopfield Networks Continued From Page 37

The C programs used to obtain solutions to the NFP and NQP are available free of charge. Contact me at the above address.

REFERENCES

Dory, R. A. (1990) Neural Networks, *Computers in Physics*, May/June, 324-328.

Hebb, D. O. (1949) "The Organization of Behaviour", Wiley, New York.

Hopfield, J. J. and D. W. Tank (1985) "Neural" computation of decisions in optimization problems, *Biol. Cybern.*, **52**, 141-152.

Hopfield, J. J. and D. W. Tank (1986) Computing with neural circuits: A model, *Science*, **233**, 625-633.

Mandziuk, J. and B. Macukow (1992) A neural network designed to solve the N-Queens Problem, *Biol. Cybern.*, **66**, 375-379.

McCracken, D.D. and W.I. Salmon (1987) "A second course in Computer Science with MODULA-2" John Wiley and Sons, Toronto.

Perelman, P. (1987) "Fun with Maths and Physics", MIR Publishers, Moscow.

Schweller, K. G. and A. L. Plagman (1989) Neural Nets and Alphabets: Introducing Students to Neural Networks, *SIGCSE Bulletin*, **21**, 3, 2-7.

Hopfield Networks Continued On Page 60*

don't complain. If you tell them that, should you like it you might buy 5 or 10, they get really friendly, and problems of a magnitude large enough will send a friendly and well-trained (also well-dressed, but that's by the way) technician to fix you right up, even though you haven't actually paid for the thing yet. On the other hand, if you decide you want to buy the gizmo card and have paid for it, problems of a sufficient magnitude will bring out a technician, still well-dressed but perhaps not as well trained, to look at the equipment for $200 an hour, with a minimum of 2 hours most of the time. And you also have to pay for his travel. And his new watch. And buy him lunch. And maybe, if the moon is aligned properly, he'll come and stare at it for an hour or two and try to fix whatever is wrong.

This is an exaggeration, of course, but you might be surprised at how close this comes to the truth sometimes. I can't say all companies are like this; there were some outstanding ones we worked with who were always very helpful, regardless of the financial status. But there are those bad ones that pop up every once in a while.

Overview

In general, I enjoyed my job a great deal. I kept busy, and I was happy with what I did (if the other people working there were happy with what I did is another question, but I like to think they were). I've learned quite a bit. I've learned some skills which I may not use again (three-dimensional TTR displays are not in high demand, or so I'm told), but of course these are specialized things and it's not right to attach undue importance to them.

On the other hand, though, what is of importance are those things you can't really put on a resume--autonomy, confidence, the ability to learn (learn quickly, on occasion), and all of those other warm glowing things which sound like an Army commercial, but really are of great importance in a job situation. They may not land you a job, but they certainly will keep you in the one you get, from my limited point of view.

All of the students I know at the SSC seem to have a lot of strength of character. I like to think that it's because of the program--we really are working on parts of projects that people all over the world are going to be concerned with, and we didn't do things which are unimportant and lead nowhere (well, sometimes we did, that's called evaluation, though, and doing evaluation work on a product that will not be used again is sort of why you do it to begin with). Overall, there's a sense (which I think is justified) that we are important, and we do contribute something tangible to the premier science project in existence.

Hopfield Networks Continued From Page 40

Shackleford J. B. (1989) Neural Data Structures: Programming with Neurons, *Hewlett-Packard J.*, June, 69-78.

Tank, D. W. and J. J. Hopfield (1987) Collective Computation in Neuronlike Circuits, *Scientific American*, **257**, 104-114.

Wallace S. R. and F. L. Wallace (1991) Two Neural Network Programming Assignments using Arrays, *Proc. Twenty-Second SIGCSE Tech. Symp. on Computer Science Education*, 43-47.