C^3P-695

Neural Networks and Dynamic Complex Systems

Geoffrey Fox, Wojtek Furmanski, Alex Ho,
Jeff Koller, Petar Simic, Isaac Wong
Caltech Concurrent Computation Program
1201 E. California Blvd.
Pasadena, California 91125
MS - 206-49
Contribution to 1989 SCS Eastern Conference, Tampa, Florida
March 28-31, 1989.

Abstract

We describe the use of neural networks for optimization and inference associated with a variety of complex systems. We show how a string formalism can be used for parallel computer decomposition, message routing and sequential optimizing compilers. We extend these ideas to a general treatment of spatial assessment and distributed artificial intelligence.

1: Introduction

We study a variety of complex systems using neural networks to address their simulation and control as well as the solution of associated optimization problems. We illustrate our methods with two broad classes of problems; routing and optimal code generation for sequential and parallel computers; trajectory determination and data fusion for distributed artificial intelligence. Our techniques are naturally highly parallel and we use concurrent computers in several of our simulations and explicit implementations.

In Sec. 2, we describe complex systems and the basic "string" or "worldline" formalism with both the neural network and path integral methods. This uses a generalized travelling salesman problem as its motivation. In Sec. 3, we discuss static, or more precisely adiabatic, complex systems and our original application to decomposition or load balancing of problems onto parallel computers. In Secs. 4 and 5, we extend these ideas to static (*neural_compiler*) and dynamic (*neural_controller*) message routing and optimal code generation for sequential computers. Combining the ideas of these sections leads to the possibility of a neural network based optimizing compiler with static or dynamic decomposition and vectorization.

In Sec. 6, we discuss a simple trajectory optimization problem involving navigation in a varied terrain. We relate this to the path integral formulation of classical dynamics. We continue these ideas in Sec. 6D with a proposal for neural network based image processing and general distributed artificial intelligence applications. We hypothesize that neural networks can be applied to both the image analysis and the situation assessment aspects of these problems.

In the final Sec. 7, we describe further research areas. These include deeper exploration of the statistical and path integral physics analogies. We also comment on a new approach to parallel event-driven simulations.

2: Complex Systems and the String Formalism

2A: Complex Systems

We have always liked the definition of complex systems given in the prospectus of the Santa Fe Institute.

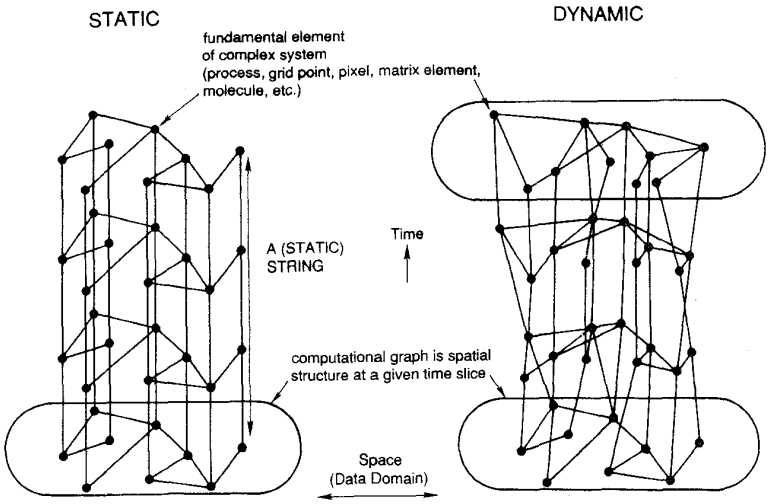
"The Need for New Options in Education and Research

The transformation of society by the scientific revolution of the 19th and 20th centuries is about to be overshadowed by even more sweeping changes arising from a growing ability to understand the complex mechanisms which are central to human concerns. The technology base of the new revolution will be provided by almost unimaginably powerful computers together with the mathematical and experimental tools and associated software which are essential to achieving an understanding of complexity. *Complex systems contain large numbers of coupled elements. The strength of the interactions between elements varies with time, space, and the nature of the surrounding environment which may also change with time.* Such systems can adapt to their environments. Examples of adaptive, complex systems include biological evolution, learning, and neural processes, intelligent computers, protein chemistry, much of pathology, and medicine, human behavior, and economics.

It is becoming increasingly evident that understanding complex systems demands mutually supportive research conducted by scholars representative of a broad spectrum of the intellectual community ranging from mathematics and the natural sciences to the humanities. Society must find new ways to nurture the necessary convergences of academic disciplines and other critical resources. Present-day academic institutions are not well designed to meet this increasingly urgent need."

In our earlier papers, we considered systems that were either static or at worst varied slowly; we called the latter adiabatic. However, central to much of this paper will be dynamic systems illustrated and contrasted with the simpler static case in Fig. 1.

We associate with any complex system a *data domain* or "*space*". If the system corresponds to a real or simulated physical system then this data domain is a typically three dimensional space. In such a simulation, the system consists of a set of objects labelled by index i and is determined by the positions $\underline{x}_i(t)$ at each time t. As shown in Fig. 1, the data domain consists of a set of interconnected nodes and this forms what we call the *computational graph*. This is defined by a time slice of the full complex system.



1. Static and dynamic complex systems defined in "Space" - "Time".

Other complex systems have more abstract data domains:

1) In a computer chess program, the data domain or "space" is the pruned tree-like structure of possible moves.
2) In matrix problems, the data domain is either a regular two dimensional grid for full matrices or a complex subset of this for sparse matrices.
3) In the parallel decomposition considered in Sec. 3, we consider the complex system formed by the parallel computer; its computational graph is formed by the nodes (memories) of the computer and the interconnection of the graph is determined by the architecture (topology) of the computer.
4) In the sequential neural compiler considered in Sec. 5, the space of the underlying complex system consists of possible locations of variables i.e., of the memory, cache, registers and CPU of the computer.

In a physical simulation, the complex system evolves with time and is specified by the nature of the computational graph at each time. If we are considering a statistical physics or Monte Carlo approach, then we no longer have a natural time associated with the simulation. Rather, the complex system is evolved iteratively or by Monte Carlo sweeps. We will find it useful to view this evolution or iteration label similarly to time in a simple time stepped simulation. We thus consider a general complex system defined by a data domain which is a structure given by its computational graph. This structure is extended in "time" to give the "space" - "time" cylinders shown in Fig. 1. In our previous examples

1) Chess: time labels depth in tree
2) Matrix Algebra: time labels iteration count in iterative algorithms or "eliminated row" in a traditional full matrix algorithm such as Gaussian elimination.
3) In the parallel decomposition, we consider static or essentially static problems with no time evolution.
4) In the *neural_compiler*, time labels clock cycles on the target computer and it labels lines of code or steps in the directed graph in the software to be mapped onto the computer.

In many areas, one is concerned with mapping one complex system into another. For instance, simulation or modelling consists of a map

(2-1)

$$\text{Nature (or system to be modelled)} \xrightarrow[\text{(theory)}]{\text{map}} \text{Idealization or Model}$$

This map would often be followed by a computer simulation which can be broken up into several maps

Model —map (numerical analysis)→ Numerical Formulation —map (user coding)→ Software (2-2)

Numerical Formulation —map (Compiler)→ Computer

Nature, the model, the numerical formulation, the software, and the computer are all complex systems. Typically one is interested in constructing the maps to satisfy certain goals such as agreement of model with effects seen in nature or running the computer simulation in a minimum time. In these cases, one gets a class of optimization problems associated with the complex systems. One recurring theme here will be the use of simulated annealing or neural network methods to address these optimization problems.

2B: The String Formalism

Consider a complex system with basic entities labelled by p. Then it is specified by the set of worldlines $\{\underline{x}_p(t)\}$ where $(\underline{x},t)$ is a point in the generalized "space" - "time" (data-domain, evolution label) associated with this system. This set of strings or paths $\{\underline{x}_p(t)\}$ are the basic degrees of freedom. In the parallel computer decomposition problem, p labels processes and $\underline{x}_p$ the processor (node) number where p is located at clock cycle t. Clearly, the execution time T_{par} of the problem represented by this collection of processes is a functional of these paths

$$T_{par} \equiv T_{par}(\{\underline{x}_0(t)\},\{\underline{x}_1(t)\}..\{x_p(t)\}..) \tag{2-3}$$

The minimization of T_{par} is our first example of an optimization problem in the string formalism and this is studied in Secs. 3 and 4.

The most straightforward approach views T_{par} as a function of the strings and a typical minimization would use Monte Carlo or simulated annealing. As shown in Fig. 2, one considers local changes in paths

$$\{\underline{x}_p(t)\} \rightarrow \{\underline{x}_p(t)\}' \tag{2-4}$$

and recovers a formalism very similar to that used in lattice gauge theories and quantum chemistry.

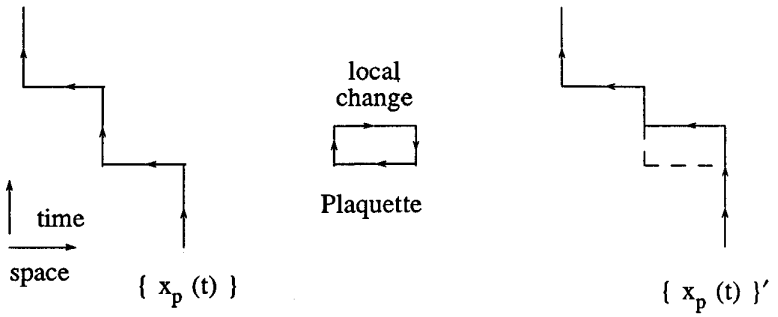
The alternative formalism uses the trick, due to Hopfield and Tank, where one introduces redundant variables $\eta_p(\underline{x},t)$ describing the string p. The binary variables η_p are defined so that

$$\eta_p(\underline{x},t) = 1 \text{ if } \textit{string } p \textit{ is at } (\underline{x},t) \tag{2-5}$$
$$= 0 \text{ if } \textit{string } p \textit{ does not pass through } (\underline{x},t)$$

For each t value, the string only passes through one $(\underline{x},t)$ value and this is enforced by adding a syntax term

$$T_{par} \rightarrow T_{par} + const. \sum_t \Big(\sum_{\underline{x}} \eta_p(\underline{x},t) - 1\Big)^2 \tag{2-6}$$

which is zero when the constraint is satisfied and otherwise positive.



2. A typical local trajectory (string) change used in a Monte Carlo approach to the string formalism. The original string $\{x_p(t)\}$ is changed to $\{x_p(t)\}'$.

Often we will extend this trick by replacing quantities like T_{par} by approximations $\tilde{T}_{par}$. Exact constraints - such as that the C.P.U. for the *neural_compiler* can only execute one instruction at a time are replaced by penalty functions that "encourage" this. The combination of the trick (2-5) and penalty functions, allows one to express complex optimization problems in a simple - often local - form with seemingly difficult constraints expressed easily if redundantly.

We will find, in several cases, two variants of the space-time minimization problem. In the most straightforward but computationally intense formulation, one solves the full optimization problem over the full space time region. Alternatively, one can use a window approach where if t_o is the current time, the state of the system at time $t_0 + 1$ is found by using a window $t_0 \le t \le t_0 + \Delta t$ (here $\Delta t > 1$ corresponds to several steps in t) with the full string dynamics. The "future" $t > t_0 + \Delta t$ is represented as an average over possibilities. In the physics analogy where objective functions like T_{par} are thought of as Hamiltonians, H, then this future average leads to external fields in H. This is the *neural_controller* approach which is computationally less intense and further is insensitive to uncertainities in future data. Of course, the reliability of this approach depends on the accuracy of the "future" average.

3: Decomposition of Adiabatic Problems

We consider the simplification of (2-2), namely the formal map

(3-1)

where we ignore important but distinct issues concerned with the intermediate steps such as software and numerical formulation. In this section, we consider cases that are either static or essentially so. Under these conditions, we are mapping the computational graph onto the computer. This load balancing or decomposition for parallel computers becomes a min-cut and equal weight graph partitioning problem. Consider for definiteness the finite element problem [Flower:87] shown in Fig. 3 where the strains in a plate require the unequal distribution of elements indicated in the figure. Suppose we wish to simulate this system on a 16 node parallel computer with a two dimensional mesh (or more generally hypercube) topology. If the elements had been uniformly distributed, then the equal area decomposition shown in Fig. 4(a) would be appropriate. However, although this would give modest and local node to node communication, it does lead to severe load imbalance shown in Fig. 4(b). We need to distribute the elements so that we minimize the appropriate sum of communication and calculation. The relative weights of these two terms depends on the characteristics of the target hardware.

Formally, one needs to minimize

$$\max_{nodes\ i} C_i \tag{3-2}$$

where C_i is the total computation time for calculation and communication. We choose to replace this mini-max problem by a least squares [Fox:88mm] minimization of

$$E = \sum_i C_i^2 \tag{3-3}$$

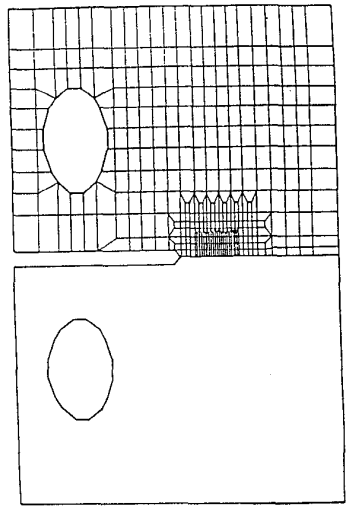
Suppose $m(m')$ label the nodal points of the computational graph. Then

$$C_i = \sum_{m \in i} [\sum_{\substack{m' \\ linked \\ to\ m}} Comm(m,m') + Calc(m)] \tag{3-4}$$

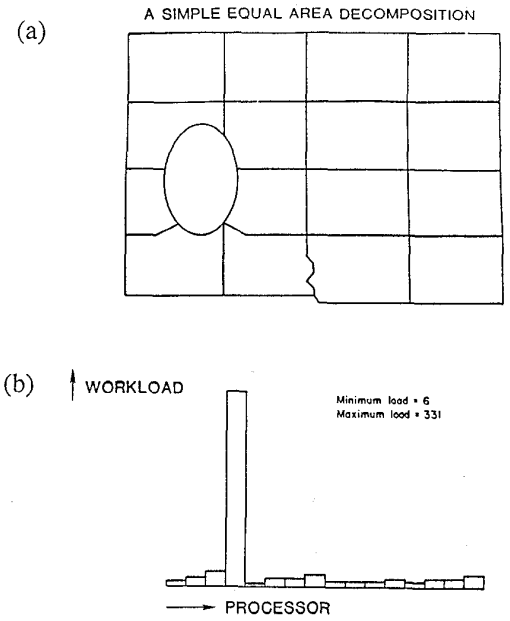
where it takes time $Calc(m)$ to simulate m and time $Comm(m', m)$ to communicate necessary information from m' to m. If we consider the case where we can neglect the quadratic communication terms, then

$$C_i^2 \approx const. \sum_{\substack{m,m' \\ for\ m\ in\ i \\ and\ m'\ linked \\ to\ m}} Comm(m,m') \tag{3-5a}$$

$$+ \sum_{\substack{m,m' \\ m,m' \\ in\ i}} Calc(m)\ Calc(m') \tag{3-5b}$$



3. A finite element mesh generated by NASTRAN. The mesh is symmetric about the horizontal division and only the top half is shown in detail.



4(a). A simple equal area decomposition of the mesh shown in Fig. 3 for a 16 node machine.

4(b). The unequal workload corresponding to distribution shown in Fig. 4(a).

The two terms in Eq. (3-5) have an interesting interpretation with a physics analogy where m, m' are particles which interact if they are linked in the computational graph. We consider these particles as moving in a space formed by the nodes and linkage of the parallel computer. Then (3-5a) is an attractive long range force which is minimized when m and m' are in the same node. It is a function of the distance in the "computer space" between m and m'; this function depends on the hardware topology and the nature of message routing on the computer. Equation (3-5b) represents a repulsive short range potential which is maximized and only nonzero when m and m' are in the same node i.

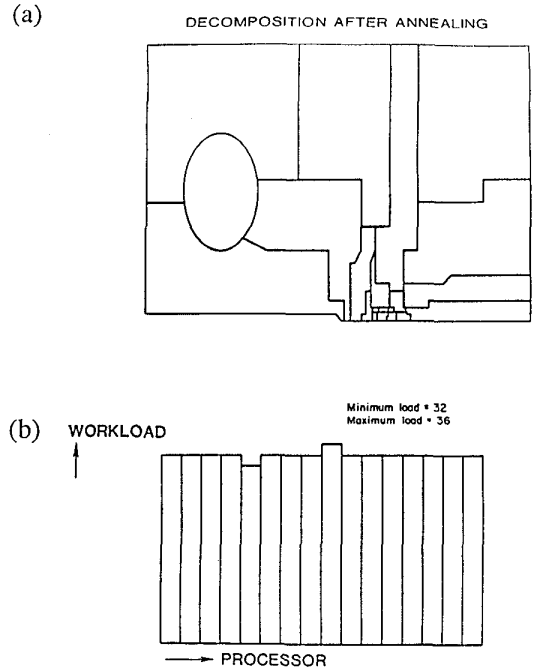
Our graph partitioning problem has been turned into one of finding the ground state of this physics system. This is formally an NP complete problem, but it is important to note that we only require an approximate solution; the use of a high level language has already compromised our performance (value of E) by a factor of two compared to optimal microcoding. We can certainly afford to "throw away" another 20% or so with an approximate as opposed to an exact minimization of E.

A natural physics approach uses the methods of statistical mechanics; this is the Monte Carlo or simulated annealing approach. It provides a convenient language for discussing time varying systems. We can view the hardware or software system which achieves the approximate minimization of Eq. (3-3) as a heat bath which equilibrates the system. If the problem is rapidly varying we can only maintain the system in equilibrium at some finite temperature T and we do not have time to find the true ground state [Fox:85a, Fox:88mm]. The techniques discussed here are appropriate for *adiabatic* systems which vary sufficiently slowly that the computational graph can be viewed as static and there is time for a reasonable minimization of Eq. (3-3) [Koller:88d].

Typical results for the problem of Figs. 3 and 4 are shown in Fig. 5. We have succeeded in dividing the mesh into roughly equal collections of nodal points; moreover each set is approximately square so as to minimize edge or communication effects.

Decomposition onto a hypercube naturally suggests a neural network formalism which is in fact quite general. Let point m reside in processor $P(m)$ and let this processor be labelled by a binary number with $\sigma_\alpha(P(m)) = \sigma_\alpha(m)$ being the α'th bit of this processor label. If we have $N = 2^d$ processors, then we associate d neural variables $\sigma_\alpha(m)$ with each point m. As described in [Fox:88e], it is straightforward to express C_i in terms of $\sigma_\alpha(m)$ and write down the Hopfield and Tank minimization formulae for (3-3) [Hopfield 86]. Using multiscale techniques, one can show that it takes time of order $M \log M \log N$ to minimize (3-3) with a neural network method on a system of size M. This is typically much faster than simulation time which is at least of order a large constant times M for a system that has an essentially constant computational graph for a large time interval - the definition of an adiabatic system. Both simulation and decomposition performance can be speeded up by a factor of order N on the parallel computer. Koller has implemented a load balancer for the iPSC/1 hypercube where simulation and balancer run as separate tasks on each node of the hypercube [Koller:88a, Koller:88d].

Originally, we had expected decomposition to be difficult but we now believe it to be quite straightforward to obtain adequate although non-optimal decompositions. As well as simulated annealing and neural networks, there are a variety of simple ad-hoc but satisfactory heuristic methods [Fox:88nn, Williams:87, Barhen:88b, Chen:88, Ercal:88, Livingston:88]. We have used these methods routinely for both finite element and particle dynamics problems [Williams:88a, Flower:87, Lyzenga:88, Williams:88d].



5(a). The "optimal" decomposition found in [Flower:87] using simulated annealing and the formalism described in Sec. 3.

5(b). The approximately equal workload for the decomposition of Fig. 5(a).

4: Decomposition of Dynamic Problems

The methods of Sec. 3 are not applicable to irregular time dependent problems such as event driven simulations and computer chess. Moreover, there is an interesting class of very regular problems which exhibit a rhythmic or cyclical computational graph for which one also needs to extend the ideas of Sec. 3. Two examples are shown in Figs. 6 and 7 for the message routing and combining switch problems. The latter is found in matrix-vector multiplication when both matrix M and vector x are distributed [Fox:88h, Fox:88e, Fox:88a]. Consider

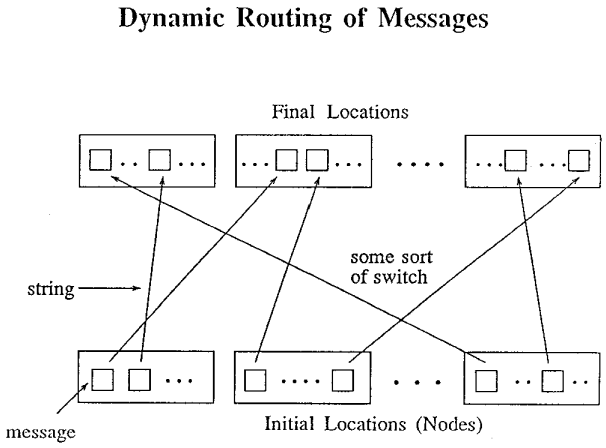
$$y_j = \sum_i M_{ji}\, x_i \tag{4-1}$$

If one sums over all values $M_{ji}\, x_i$ in each node I, then this becomes

$$Y_j = \sum_I y_j^{(I)}$$

or the accumulation of many (= number of components of y) global sums. The analytic solution of this is known as the algorithm *fold* in [Fox:86e] and illustrated in Fig. 7.

In the physics analogy of Sec. 3, one might consider the separate $y_j^{(I)}$ as particles. However, these must move in a correlated fashion through the nodes of the computer in a way such that $y_j^{(I_1)}$ and $y_j^{(I_2)}$ are combined (added) when they "collide" at a common node "on the way" from (I_1) and (I_2) to the destination node J containing y_j. The physics analogy is incomplete as we have an instantaneous energy function but no equations of motion. Rather, we use the worldline formalism introduced in Sec. 2 and consider as degrees of freedom the complete time dependent strings which terminate at one end on y_j in J and at the other end on $y_j^{(I)}$. We must drape these strings on the computer nodes so as to minimize the total time. In a traditional physics problem, one can use either a path integral or equations of motion formulation. Here only the former seems possible and one must regard the paths and not the instantaneous particles as the basic degrees of freedom.



Neural_Router dynamically routes messages given current message location and destination

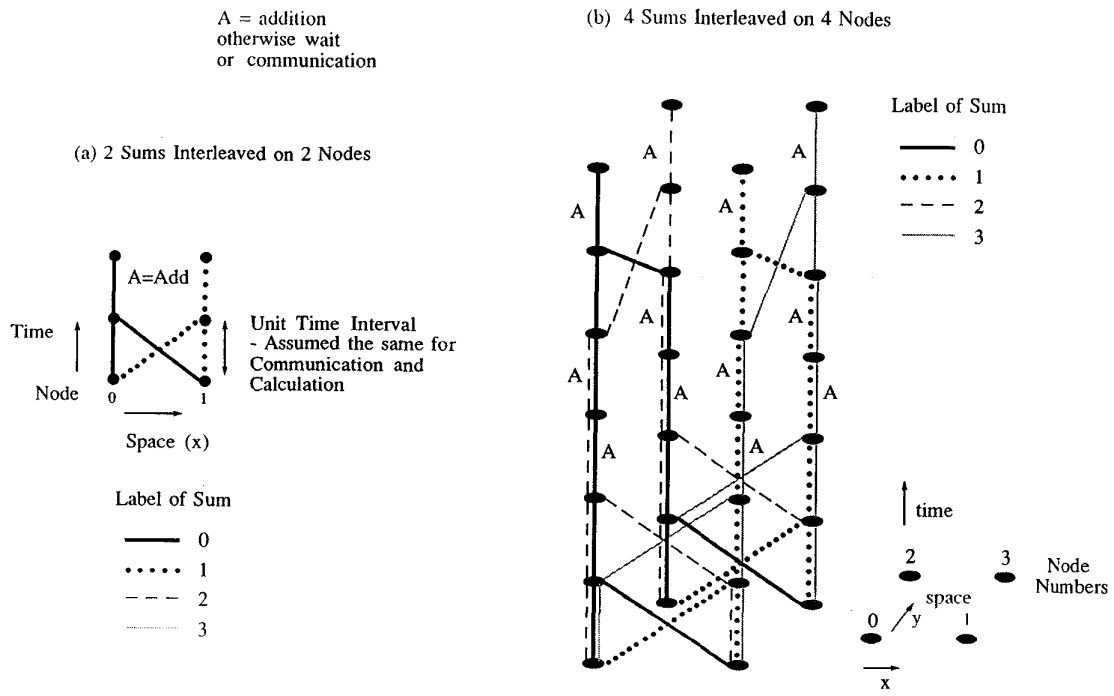
6. A message routing schematic of the problem addressed by the *neural_router*.

In [Fox:87a, Fox:88f], initial results are presented for this combining switch using neural networks for the controller or "window" formulation mentioned at the end of Sec. 2. The neural networks include terms that

- Attract strings corresponding to some sum y_j.
- Repel strings corresponding to different sums y_{j_1} and y_{j_2}.
- Attract strings to destination J containing y_j.

Typical results are shown in Fig. 8 for our implementation called the *neural_accumulator*. We considered 16 sums accumulated on 16 nodes for a sparse matrix M such that only a fraction f $(0 \le f \le 1)$ of nodes contain a contribution $M_{ji}x_i$ for each y_j. The case $f = 1$ is solved exactly by the analytic *fold* algorithm but the case $f < 1$ only has an approximate deterministic solution called the *crystal_accumulator* [Fox:88g, Fox:88a].

Hopfield and Tank originally illustrated the neural network approach to the simple travelling salesman (TSP) problem. One minimizes the total travel time of a single salesman who must visit each of M cities once. Our path formalism can be viewed as a multiple travelling salesman method. In the original TSP, one has a single path or string to be routed (draped) over all the cities. In the new formalism, we have several interacting salesmen.



7. The combining switch illustrated for two and four nodes. The *fold* algorithm is known analytically in this case [Fox:88h, Fox:88a] and shown in the figures.

5: Optimizing Compilers

In the previous section, we considered salesmen which were messages or processes moving between computers. Here we focus on a single node and consider salesmen as variables moving between memories (registers, cache, main memory, paged out memory...) and C.P.U. of a single computer. This leads us to consider our formalism for optimizing compilers. Let us illustrate our ideas with the problem of producing code for the simple C program:

$$z = z * (x + y) - y \tag{5-1}$$

As shown in Fig. 9(a), this is represented by a directed acyclic graph (dag) where we will label nodes and leaves of the dag by an index i. We will consider the evaluation of (5-1) on a very idealized computer with a single register on which all arithmetic operations are performed. Of course, the solution of this code generation problem is "obvious" by inspection but it is sufficient to illustrate our neural network approach.

We let m label the registers and memory locations in the machine, t label the clock cycle, and introduce neural variables $\eta\ (m,i,t)$ to indicate whether quantity i is in location m at cycle t. The code generation problem is quite similar to the routing problem of the previous section: we construct an energy function containing syntax terms to ensure that

1) Code correctly represents the program (e.g., t1 and t2 in Fig. 9(b) are created before being used),

2) Fixed-t configurations represent possible machine states (i.e., one quantity per storage location), and

3) States at consecutive times are related by a machine operation.

We also know the values of the initial machine state $\eta\ (m,i,0)$ and the desired final machine state $\eta\ (m,i,T)$ for some appropriate T. We add terms to the energy to minimize the number of intermediate machine states, and use Hopfield Tank style neural network evolution equations to find a minimum of this energy.

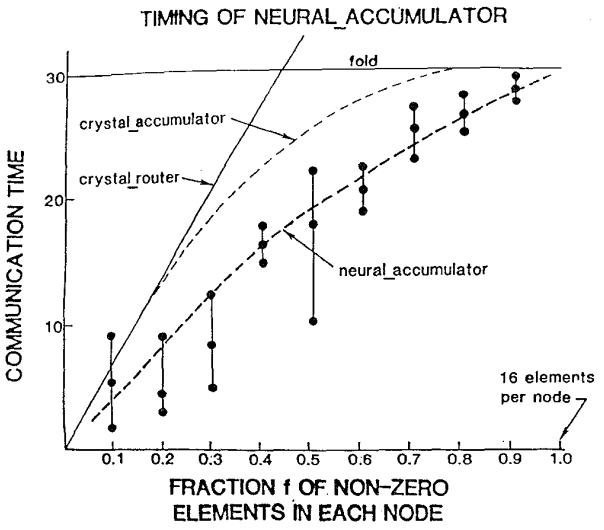
We have found a convenient way to build syntactic terms for even complicated constraints of the type 1,2,3) above. Since we ultimately interpret the neural variable η as logical variables with 1 = TRUE and 0 = FALSE, any constraint is some logical statement involving the $\eta\ (m, i, t)$, say $P\ (\eta_1, \eta_2,...)$ = TRUE. Extend the logical operations to fractional values of η via $a \wedge b \rightarrow \underline{ab}$, $a \vee b \rightarrow a + b - ab$, $\overline{a} \rightarrow 1 - a$. Then we penalize violations of this constraint by adding a multiple of $\overline{P}$ to the energy. In the simple study reported in [Wilson:88], an appropriate coefficient was found by trial and error.

A typical example corresponds to constraint 2) above. Suppose $\eta\ (m, i, t) = 1$ indicates that memory location m contains quantity i. Then we cannot store any other variables in this location, i.e., $\eta\ (m, j, t)$ must be zero for all $j \neq i$. This constraint corresponds to a term $\overline{P}$ in the energy function E of form

$$E = \overline{P} = \eta\ (m, i, t)\ (\sum_{i \neq j} \eta(m, j, t)) \tag{5-2}$$

i.e., if $\eta(m, j, t) = 1$, we require all $\eta(m, j \neq i, t) = 0$. More generally, we have constraints of the form

$$A\ (\eta_1, \eta_2...)\ \textit{requires}\ B\ (\eta_1, \eta_2...) \tag{5-3}$$



8. Results from the *neural_accumulator* described in Sec. 4 which addresses the dynamic irregular version of the problem shown in Fig. 7.

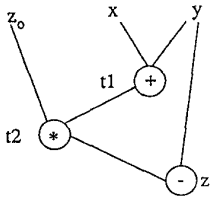
(a) Very Simple Expression

e.g. C : `z = z * (x+y) - y`

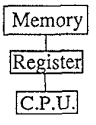
Intermediate C :

```
t1 = x + y
t2 = z * t1
 z = t2 - y
```

Directed Acyclic Graph (dag) :



(b) Very Simple Model Computer



One - register machine, with operations

LOAD M	R ← M
STORE M	M ← R
OP M	R ← OP M
OP R, M	R ← R OP M

9(a). A simple line of code and its representation by a directed acyclic graph.

9(b). The simple model computer discussed in Sec. 5.

where A and B are logical expressions i.e., functions of the $\eta(m, i, t)$. The constraint (5-3) corresponds to

$$P = B \vee \bar{A} \tag{5-4}$$

and a term $\bar{P} = A \wedge \bar{B} = A\bar{B}$ in the energy function.

In [Fox:88e], we showed that adding a noise term to the Hopfield Tank equations improved the performance of the resultant network for the problems of Sec. 3. We have used the same "bold network" for code generation and the results reported in [Fox:88cc] are encouraging. However, there is an important issue we need to address before a practical system could be produced. We do not need an exact minimum of the term in E corresponding to the execution time of the code. However, many of the syntax constraints must be exactly satisfied or else the code will lead to incorrect results. Our current plan is to allow small syntax violations, and use a postprocessor to repair them. This is a promising alternative to the all-or-nothing conventional approach to neural network optimization. It also provides a natural way to normalize the penalty terms in the energy: the penalty for a syntax violation should equal the extra execution time the postprocessor would have to add to the program to fix up the error. Note that the postprocessor could also be a neural network. This idea applies equally to the original travelling salesman neural network formulation of Hopfield and Tank, and we are currently testing its effectiveness there.

The neural network approach to optimizing compilers has several attractive features:

1) As our approach explicitly minimizes an analytic function, it is possible to systematically improve any solution - perhaps by using simulated annealing. This would allow one to adjust the compile time and optimality of code according to ones needs. A long extensive optimization would be appropriate before a 6000 hour CRAY run on a "grand challenge"; a quick non-optimal option would be appropriate when debugging.
2) This method naturally incorporates the "exact" architecture of the computer in the detailed form of E. In particular, it should in principle be able to handle complex memory hierarchies, which are present in high performance computers (such as the CRAY-2) but hard to handle with conventional techniques.
3) One should be able to build rather portable compilers with this technique. The manufacturer of a new RISC architecture multi-function superchip need only specify the particular form of E to allow a portable neural network based compiler to be used.

The key to the viability of our new approach to computers is the performance of optimizing neural networks with large numbers of neurons. As pointed out in [Wilson:88], the initial naive networks break down on large systems. We expect that careful construction of hierarchical systems and the use of more powerful optimization strategies will lead to much better performance on large systems. However, this research is still in its infancy and we cannot make a firm prognosis.

6: Distributed Artificial Intelligence

6A: Introduction

We believe that neural networks and the string formalism are the natural approach to many distributed artificial intelligence problems. An essential feature of the string method is that it preserves properly the underlying structure of the "space" - "time" involved. When this is a physical world, the constraints of causality (finite signal and vehicle velocities) and terrain structure are important and difficult to handle except by some analytic cost functions as in the string approach. An expert system approach would be quite appropriate when the underlying complex system was amorphous and without analytic structure; this would perhaps be the case if the underlying "space" is that formed by a set of perceptions or ideas. We are developing a neural network approach to decision making where the information is spatially and temporally labelled i.e., where the complex system is based on a physical world. Some initial results for navigation are given in Sec. 6B while the following section describes an application to video games. A more general application to image interpretation is sketched in 6D.

6B: The Neural Navigator

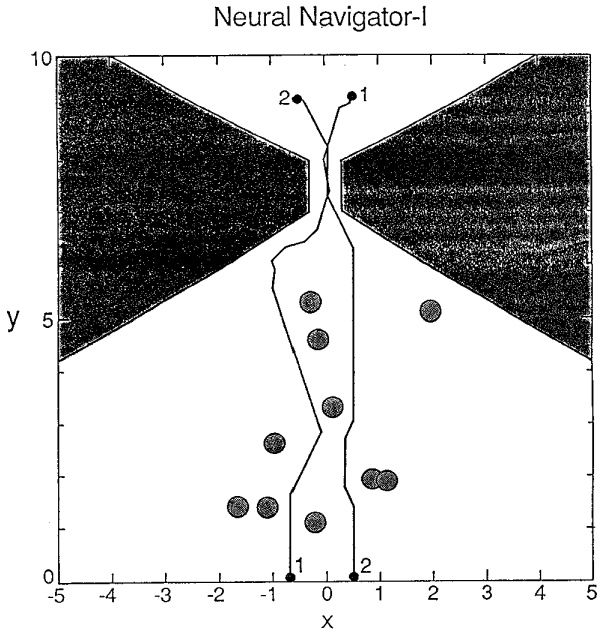
Our ideas are quite general but for definiteness, consider an optimal path problem where we have a collection of objects, which we call vehicles, in a two dimensional space. We wish to navigate the vehicles from initial starting positions to final destinations so as to minimize the travel time. In the following, we consider just two vehicles labelled by i=1, 2. The essential idea is to again view the paths as the degrees of freedom and use the redundant neural variables $\eta_i(\underline{x}, t)$ to parameterize these paths. We again need to form an energy functional $E(\{\eta_1\}, \{\eta_2\})$ which incorporates both the goal (minimal travel time) and the various constraints. Perhaps the problem is best illustrated by typical results shown in Figs. 10 and 11. We have the two vehicles starting at the bottom of the figure and reaching destinations at the top. They must navigate so as to avoid each other and respect the terrain constraints. In this case, the latter corresponds to a collection of hills (rocks) with sharp boundaries i.e., the shaded areas are to be avoided. These hills consist of several randomly placed rocks and a major "range" with only a narrow passable region. Our energy function E has several terms

$$E = \sum_{j=0}^{5} A_j E_j \tag{6-1}$$

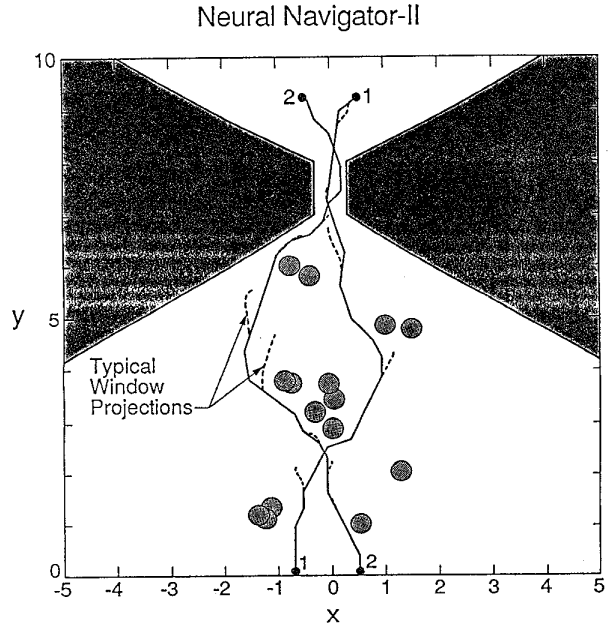
with variable coefficients A_j reflecting the "importance" of the constraint. Let us describe these terms qualitatively.

We will do this in the context of the window philosophy explained in Sec. 2. We have a starting time t_0 and only use the neural variables $\eta_i(\underline{x},t)$ in the window $t_0 \le t \le t_0 + \Delta t$ as the dynamical variables. The future $t > t_0 + \Delta t$ is, in principle, represented as an average over paths but in practice is approximated intuitively.

0) The first term represents the goal of minimal travel time. Let us just note that we can go back and forth between a neural and conventional space time description with the simple equations.



10. An example of the *neural_navigator* for the motion of two vehicles (1,2) from the bottom to the top of the figure.



11. A second example of the *neural_navigator* with rather more complex terrain than in Fig. 10. Shaded areas (rocks, hills) are to be avoided. Shown are typical window projections discussed in Sec. 6B.

$$positions = \underline{x}_i(t) = \sum_{\underline{y}} \underline{y}\,\eta_i(\underline{y}, t) \tag{6-2}$$

$$\underline{v}_i(t) = velocity = \frac{d\underline{x}_i(t)}{dt} = \sum_{\underline{y}} \underline{y}\,(\eta_i(\underline{y}, t+\delta t) - \eta_i(\underline{y}, t))/\delta t \tag{6-3}$$

Let $\underline{d}_i$ be the final destination of the i'th vehicle. Then we express the goal of reaching the destination by

$$E_0 = A_0 \sum_{\substack{i \\ over\ vehicles}} \sum_{\substack{t \\ over\ window}} [\,|\underline{x}_i(t+1) - \underline{x}_i(t)| + |\underline{x}_i(t+1) - \underline{d}_i| - |\underline{x}_i(t) - \underline{d}_i|\,] \tag{6-4}$$

In the examples of Figs. 10 and 11, this simple form is an adequate "average of the future"; it would not be sufficient if, for instance, the narrow pass was displaced (in x) from the destinations. We are considering a general multiscale method in space and time to provide a generally accurate estimate of E_0. The whole problem is first solved with a coarse space and time grid and this crude solution is used to estimate the goal constraint (6-4). We have very good experience with multiscale methods in space [Furmanski:88c , Battiti:88b] but need to extend them to allow variable temporal scales.

1) The second term in Eq. (6-1) expresses the smoothness of the trajectories and is taken as

$$E_1 = A_1 \sum_{\substack{i \\ over\ vehicles}} \sum_{\substack{t \\ window}} (d^2\underline{x}_i/dt^2)^2 \tag{6-5}$$

2) The third term ensures that the vehicles keep a reasonable distance apart and in the case of two vehicles we use

$$E_2 = A_2 \sum_{\underline{x}_1\,\underline{x}_2} \sum_{\substack{t_1\,t_2 \\ in\ window}} \exp(-|t_1 - t_2|). \exp(-|\underline{x}_1 - \underline{x}_2|^2/\sigma^2)\,\eta_1(\underline{x}_1, t_1)\,\eta_2(\underline{x}_2, t_2) \tag{6-6}$$

where σ is a suitable distance scale and we actually cut off sums and keep only those $\underline{x}_i$, t_i where vehicles are close.

3) In the current model calculations, the terrain constraint comes in two terms. In the first we represent hills and rocks by a function $H(\underline{x})$ which is zero on the level and unity in forbidden regions occupied by the hills. Then we constrain the vehicles to passable regions by the constraint

$$E_3 = A_3 \sum_{\substack{i \\ over\ vehicles}} \sum_{\substack{t \\ over\ window}} \sum_{\underline{x}} H(\underline{x})\,\eta_i(\underline{x}, t) \tag{6-7}$$

4) We also incorporate a maximum velocity $\underline{v}_i^{max}(\underline{x})$ by the constraint

$$E_4 = A_4 \sum_{\substack{i \\ over\ vehicles}} \sum_{\substack{t \\ over\ window}} \sum_{\underline{x}} \eta_i(\underline{x}, t)\,\Theta\,(\underline{v}_i^{max}(\underline{x})^2 - \underline{v}_i^2(t)) \tag{6-8}$$

where $\underline{v}_i$ is calculated as in (6-3); actually this is not accurate and we remove "jitter" by averaging not over two but many time intervals in Eq. (6-3). This is appropriate as we choose the grid so that a vehicle moves about two space grid positions in a single time step. In Eq. (6-8), Θ is any reasonable function, such as the Heaviside function, that is zero when its argument is negative.

In the examples we use v_i^{max} as a constant independent of i. However, it is clear how we can use a vehicle (i) and position ($\underline{x}$) dependent velocity. This would allow sophisticated knowledge of the terrain to be included so as to distinguish roads, fields, forests, etc.

5) Finally, we have syntax constraints where we use

$$E_5 = A_5 \sum_{\substack{i \\ over\ vehicles}} \sum_{\substack{t \\ over\ window}} (\sum_{\underline{x}} \eta_i(\underline{x}, t) - 1)^2 \tag{6-9}$$

The results of using conventional Hopfield Tank minimization (without the bold network or other improvements mentioned in Sec. 5) are shown in Figs. 10 and 11.

In Fig. 11, we show not only the complete paths but also a few "window" projections. The path is constructed as a collection of window predictions; each window had $\Delta t = 5$ and we only took the predictions for just the first two time steps in constructing the path. The figures show the last three (ignored) steps for sample starting positions.

Clearly, our examples, methods, and understanding are very rudimentary. However, we believe that the initial results are encouraging. Neural networks can be used with the string formalism to project into the future. Pretentiously, we could call this "neural thought".

6C: A Videogame

One amusing application of the *neural_navigator* is to the control of videogames. Currently a sizeable part of teenage America spends many hours each day at a joystick controlling Pacman and spaceships as these cardboard characters journey through an idealized world fighting the good fight. Our idea is to use the neural network approach sketched in the previous section to control such games. Thus, our youth will use their joysticks to vary the coefficients A_i in Eq. (6-1) rather than to directly control the motion of characters. We have started the development of such a game using a parallel computer, the NCUBE hypercube, as the host. We initially exploit the hardware parallelism with a functional decomposition; each node controls a different object. However, the neural networks developed in Sec. 6A allow massive parallelism and the parallel computer is a natural host. We choose the NCUBE because of its real time parallel graphics interface allowing high performance update of the game display.

We have implemented on the NCUBE a 3D Asteroids video game [Ho:88k]. The Asteroids game features a battle of spacecraft in a 3D toroidal space with inert meteorites of various sizes. The game supports multi-players and mixed communication protocols. It can be played either in interactive or batch mode. In interactive mode a player can maneuver a spacecraft by keyboard or graphics tablet control. 3D visual display of the game uses the NCUBE Real-Time Parallel Graphics Board which has 16 NCUBE processors and a Hitachi HD63484 drawing/video chip.

The 3D "Asteroids" type of game is chosen for our initial implementation because of its simplicity. In this game, there are player spacecraft(s), missiles, and inert rocks of various sizes, which obey physical laws such as conservation of momentum and energy. Currently, spacecraft-maneuvering includes turn, yaw, thrust forward and backward, and missile firing. Customized maneuver and operational capabilities can be added to different classes of spacecraft when advanced functionalities are needed. Large rocks split into multiple smaller rocks when they collide with other objects or when they are hit by missiles. On the other hand, when player spacecraft are hit by rocks, missiles, or collide with other spacecraft, they lose "energy" according to a pre-determined magnitude. When a spacecraft's energy is exhausted, it is out of the game.

The implementation of the game allows multiple players (spacecraft.) New players can be added at any time of the game. Spacecraft can be controlled either interactively by human players or by computer programs or controllers. Controllers for player spacecraft are isolated from the rest of the game. These controllers can be written in CrOS III, Cubix [Fox:88a], or VERTEX, the NCUBE commercial software system. The high performance CrOS III software will allow excellent parallel implementation of the neural network controllers discussed in Sec. 6B.

The design of the Asteroids videogame is highly modular in order to allow rating of novel controllers and there are three types of programs involved. They are the "game driver", "player", and "graphics driver" programs.

There is only one game driver program running on a subcube of the main NCUBE array at all times of the game. The game driver implements the rules of the game, i.e., it evolves game objects forward in time.

There is one player program for each player in the game, each running on a distinct subcube of the NCUBE array. These programs implement the different strategies for computer-controlled players. At each time step, each player program (controller) receives information on the current state of the game from the game driver. The player program can also send "moves" such as "start thrusting" or "fire missile" back to the game driver. The player program can also be written so that it implements an interface for an interactive human player. The program generates a 3-D perspective graphics display based on the received state information, gets input from the human player, interprets input as relevant "moves", and sends them to the game driver. Currently, all players will receive the same amount of information from the game driver. Future improvements will differentiate classes of spacecraft. Class-sensitive information will be sent from the game driver to the controllers.

The graphics driver is a program that runs on the NCUBE Real-Time Parallel Graphics Board. This board has 16 processors. An identical copy of the graphics driver program runs in parallel on the 16 processors. The graphics driver performs more than low-level graphics operations. It also handles message routing. When the game driver sends a message to a player and vice versa, the message is routed through the graphics board, the graphics driver program loaded there recognizes that the message is meant for a player, and thus performs a message forwarding from the graphics board to the appropriate destination, or subcube, in the NCUBE array.

Currently, we have the basic system running but we are still experimenting with the design of an appropriate *neural_navigator*. We expect that it could be another year before a realistic neural network controller of the style of Sec. 6B could be implemented. Clearly, the treatment of asteroids is similar to that of terrain constraints (rocks) in the examples of Fig. 10 and 11.

6D: Spatial Decision Making

We are currently developing a sophisticated neural network based image analysis (computer vision) system [Furmanski:88c, Furmanski:88e]. This can be combined with the ideas of Sec. 6B to produce an elegant neural network based system that will analyze and interpret general images. By image analysis, we mean the processing of sensory data to form a list of objects and their spatial locations. This processing involves geometric and noise cleanup, edge detection, stereo, optical flow, shape from shading and texture classification. We have built our system incorporating multiscale ideas both because we expect the need to fuse data at different resolutions and also because multiscale algorithms greatly outperform conventional ones [Terzopoulos:83, Terzopoulos:86] For instance, conventional relaxation on a $\sqrt{N} \times \sqrt{N}$ pixel image takes time of order $N^{3/2}$; multiscale takes time of order $N \log N$ [Battiti:88b]. This image analysis can be viewed as producing the necessary information to allow the neural-navigator to predict the future i.e., it would provide the data needed in Eqs. (6-4) to (6-10). We are currently designing such an integrated system. We expect useful feedback to occur between the prediction and analysis sections. We also are extending the spatial multiscale methods to the temporal domain as mentioned in Sec. 6B.

7: Some Future Directions

Here we summarize three areas where we are extending some of the methods presented earlier.

7A: Classical Dynamics

The neural network approach of Sec. 6B was presented for a rather ad hoc optimal path problem. However, it can be applied directly to find exact classical trajectories. Consider a one dimensional pendulum with Hamiltonian

$$H = T + V \tag{7-1}$$

$$T = 1/2m\dot{x}^2 \tag{7-2}$$

$$V = 1/2kx^2 \tag{7-3}$$

Then we can either find trajectories by the conventional Newton's law

$$m\ddot{x} = -kx \tag{7-4}$$

or by the equivalent path integral formalism

$$\delta \int_{t_0}^{t_1} (T-V)dt \tag{7-5}$$

where we need to find the minimum (stationary value) of a Lagrangian $L = T-V$. After discretizing space and time, one finds exactly the problem of Sec. 6B with the energy E in Eq. (6-4) replaced by

$$L = 1/2m\dot{x}^2 - 1/2kx^2 \tag{7-6}$$

We introduce neural variables $\eta(x, t)$ as before, and write $\dot{x}$ in terms of η using Eq. (6-3).

We are currently investigating this problem as it is clearly one for which we have substantial intuition. We note that the neural network path integral formalism is much more computationally complex than direct integrations. However, it can be implemented efficiently on parallel machines; in particular, the SIMD Connection Machine or special purpose neural network hardware. Thus, in a future world dominated by parallel machines, such path integral formalisms could be attractive compared to the direct sequential method of Eq. (7-4).

In Sec. 6B, we used a separated neural variable $\eta_i(x,t)$ for each vehicle i. If the vehicles were of identical type, it would be natural to use a single neural field $\eta(x,t)$ representing the vehicle density. At each time instance t, one requires a given number N of the $\eta(x,t)$ to be one and the rest zero. One would replace (6-10) by

$$E_5 = A_5 \sum_t (\sum_x \eta(x, t)-N)^2 \tag{7-7}$$

In this way one could, for instance, solve several (N) pendula problems (with different initial conditions) simultaneously. We hope to investigate this idea in the near future.

7B: Event Driven Simulation

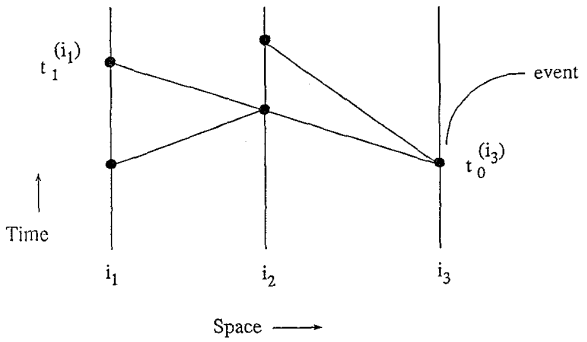
The string formalism is the natural description of an event driven simulation. As shown in Fig. 12, we have several world lines $i_1\ i_2\ i_3$... interacting by events at discrete variable times $t_j^{(i_k)}$. Nature is time stepped and a time driven (synchronized) simulation is the obvious formulation of a physical simulation. However, this can be very inefficient if the world is modelled as many macroscopic objects interacting at variable irregular times. We note that such an event driven description of nature is usually inherently inexact as we ignore the possibility of other events at intermediate times. For instance, suppose we model a game of billiards where the world lines are trajectories of balls and events are collisions. In a time stepped approach we are guaranteed "exact" results as long as the time step is small enough; we cannot miss a collision. In an event driven approach, we can take long, ball dependent, time steps between collisions; this is much faster as long as we catch all collisions.

The time stepped approach is exact, easily parallelized (the problem is loosely synchronous in the language of [Fox:88b] [Fox:88a]) but computationally complex. The event driven approach is faster on a sequential machine but hard to parallelize (asynchronous according to [Fox:88b]). As the strings interact irregularly, they cannot easily be evolved in parallel [Jefferson:88]. However, these problems in parallel simulation are only present if one insists on exactly reproducing the sequential simulation. As the original event driven formulation is in-trinsically inexact, it seems natural to use an approximate simulation method. This could be either the neural network method, as described in Sec. 6B, or simulated annealing. In the latter case, by varying the annealing temperature one can control the precision of the parallel simulation.

This approach to event driven simulation was suggested to us by Dan Weiner and we are currently looking for some good examples to test our ideas on. We need a case where we understand both the event and time driven approaches so that we can quantify the precision of the event driven approach.

7C: Neural Networks and Statistical Physics

Simic has extended the well known analogy between the Hopfield Tank neural networks and statistical physics. The usual neural network formulae follow from the mean field approximation to the analogous physics problem. They can be obtained from a variational principle from the thermodynamic free energy calculated from the semi-classical or tree level approximation. Now, clearly this does not capture all the information in the physics formulation, in particular statistical correlations due to fluctuations, and it is interesting to explore more sophisticated approximations. Simic has calculated the one loop contribution to the free thermodynamic energy for a range of spin glass like neural networks. Expanding in powers of the coupling, J_{lm}, he finds that the first term is of the same order of magnitude as the tree level term, and in particular it exactly cancels the extra term introduced by Fox and Furmanski in the bold network [Fox:88e]. However, the next term in the expansion is of the same sign and similar functionality, and as we have described in Sec. 5, it will tend to flip the value of each spin improving the convergence of the network. This is interesting not only as a theoretical justification of the bold network but as an illustration of the power of the physics analogy. We are continuing this research clarifying the relation between neural networks, simulated annealing, statistical physics and their use in optimization and learning [Simic:88a,b].



12. Three world lines of an event driven simulation showing typical events at times $t_j^{(i_j)}$.

References

[barhen:88] Barhen, J., Gulati, S., and Iyengar, S. S. "The pebble crunching model for load balancing in concurrent hypercube ensembles," in G. C. Fox, editor, *The Third Conference on Hypercube Concurrent Computers and Applications, Volume 1*, pages 189–199. ACM Press, 11 West 42nd Street, New York, NY 10036, January 1988. Caltech Report C3P-610.

[battiti:88b] Battiti, R. "Surface reconstruction and discontinuity detection: A hierarchical approach." Technical Report C3P-676, California Institute of Technology, October 1988.

[chenw:88] Chen, W. K., and Gehringer, E. F. "A graph-oriented mapping strategy for a hypercube," in G. C. Fox, editor, *The Third Conference on Hypercube Concurrent Computers and Applications, Volume 1*, pages 200–209. ACM Press, 11 West 42nd Street, New York, NY 10036, January 1988.

[ercal:88] Ercal, F., Ramanujam, J., and Sadayappan, P. "Task allocation onto a hypercube by recursive mincut bipartitioning," in G. C. Fox, editor, *The Third Conference on Hypercube Concurrent Computers and Applications, Volume 1*, pages 210–221. ACM Press, 11 West 42nd Street, New York, NY 10036, January 1988.

[flower:87a] Flower, J., Otto, S., and Salama, M. "Optimal mapping of irregular finite element domains to parallel processors." Technical Report C3P-292b, California Institute of Technology, August 1987. in Proceedings, Symposium on Parallel Computations and Their Impact on Mechanics, ASME Winter Meeting, Dec. 14–16, Boston, Mass.

[fox:85a] Fox, G., Otto, S. W., and Umland, E. A. "Monte Carlo physics on a concurrent processor," *Journal of Statistical Physics*, 43(5/6), June 1986. Proceedings of the Conference on Frontiers of Quantum Monte Carlo, September 3–6, 1985 at Los Alamos. Caltech Report C3P-214.

[fox:86e] Fox, G. "Iterative full matrix-vector multiplication on the hypercube." Technical Report C3P-336, California Institute of Technology, 1986.

[fox:87a] Fox, G. C., and Furmanski, W. "The physical structure of concurrent problems and concurrent computers." Technical Report C3P-493, California Institute of Technology, August 1987.

[fox:88a] Fox, G. C., Johnson, M. A., Lyzenga, G. A., Otto, S. W., Salmon, J. K., and Walker, D. W. *Solving Problems on Concurrent Processors*, volume 1. Prentice-Hall, Inc., Englewood Cliffs, NJ 07632, 1988.

[fox:88b] Fox, G. C. "What have we learnt from using real parallel machines to solve real problems?," in G. C. Fox, editor, *The Third Conference on Hypercube Concurrent Computers and Applications, Volume 2*, pages 897–955. ACM Press, 11 West 42nd Street, New York, NY 10036, January 1988. Caltech Report C3P-522.

[fox:88cc] Fox, G. C., and Koller, J. G. "Code generation by a neural network — general principles and elementary examples." Technical Report C3P-650, California Institute of Technology, July 1988. To be published in Journal of Distributed and Parallel Computing.

[fox:88c] Fox, G. C., editor. *The Third Conference on Hypercube Concurrent Computers and Applications.* Jet Propulsion Laboratory of the California Institute of Technology, ACM Press, 11 West 42nd Street, New York, NY 10036, January 1988. Volume 1 - Architecture, Software, Computer Systems and General Issues; Volume 2 - Applications.

[fox:88e] Fox, G. C., and Furmanski, W. "Load balancing loosely synchronous problems with a neural network," in G. C. Fox, editor, *The Third Conference on Hypercube Concurrent Computers and Applications, Volume 1*, pages 241–278. ACM Press, 11 West 42nd Street, New York, NY 10036, January 1988. Caltech Report C3P-363b.

[fox:88f] Fox, G. C., and Furmanski, W. "A string theory for time dependent complex systems and its application to automatic decomposition," in G. C. Fox, editor, *The Third Conference on Hypercube Concurrent Computers and Applications, Volume 1*, pages 285–305. ACM Press, 11 West 42nd Street, New York, NY 10036, January 1988. Caltech Report C3P-521.

[fox:88g] Fox, G. C., and Furmanski, W. "Hypercube algorithms for neural network simulation the Crystal_Accumulator and the Crystal_Router," in G. C. Fox, editor, *The Third Conference on Hypercube Concurrent Computers and Applications, Volume 1*, pages 714–724. ACM Press, 11 West 42nd Street, New York, NY 10036, January 1988. Caltech Report C3P-405b.

[fox:88h] Fox, G. C., and Furmanski, W. "Optimal communication algorithms for regular decompositions on the hypercube," in G. C. Fox, editor, *The Third Conference on Hypercube Concurrent Computers and Applications, Volume 1*, pages 648–713. ACM Press, 11 West 42nd Street, New York, NY 10036, January 1988. Caltech Report C3P-314b.

[fox:88mm] Fox, G. C. "A review of automatic load balancing and decomposition methods for the hypercube," in M. Schultz, editor, *Numerical Algorithms for Modern Parallel Computer Architectures*, pages 63–76. Springer-Verlag, 1988. Caltech Report C3P-385.

[fox:88nn] Fox, G. C. "A graphical approach to load balancing and sparse matrix vector multiplication on the hypercube," in M. Schultz, editor, *Numerical Algorithms for Modern Parallel Computer Architectures*, pages 37–62. Springer-Verlag, 1988. Caltech Report C3P-327b.

[furmanski:88c] Furmanski, W., and Fox, G. C. "Integrated vision project on the computer network," in E. Clementi and S. Chin, editors, *Biological and Artificial Intelligence Systems*, pages 509–527. ESCOM Science Publishers B.V., P. O. Box 214, 2300 AE Leiden, The Netherlands, 1988. Caltech Report C3P-623.

[furmanski:88e] Furmanski, W. "Intelligence engineering." Technical Report C3P-701, California Institute of Technology, 1988.

[ho:88h] Ho, A., Fox, G. C., Snyder, S., Chu, D., and Mlynar, T. "Parallel 3D asteroids, a status report." Technical Report C3P-681, California Institute of Technology, November 1988. Summer Undergraduate (SURF) Report.

[hopfield:86] Hopfield, J. J., and Tank, D. W. "Computing with neural circuits: a model," *Science*, 233:625, 1986.

[jefferson:88] Jefferson, D., Beckman, B., Blume, L., Diloreto, M., Hontalas, P., Reiher, P., Sturdevant, K., Tupman, J., Wedel, J., Wieland, F., and Younger, H. "The status of the Time Warp operating system," in G. C. Fox, editor, *The Third Conference on Hypercube Concurrent Computers and Applications, Volume 1*, pages 738–744. ACM Press, 11 West 42nd Street, New York, NY 10036, January 1988. Caltech Report C3P-627.

[koller:88a] Koller, J. "A dynamic load balancer on the Intel hypercube," in G. C. Fox, editor, *The Third Conference on Hypercube Concurrent Computers and Applications, Volume 1*, pages 279–284. ACM Press, 11 West 42nd Street, New York, NY 10036, January 1988. Caltech Report C3P-497.

[koller:88d] Koller, J., Fox, G. C., and Furmanski, W. "Physical optimization and dynamic load balancing." Technical Report C3P-670, California Institute of Technology, October 1988. Submitted to IEEE Conference, Newport Beach, CA — held in June 1989.

[livingston:88] Livingston, M., and Stout, Q. F. "Distributing resources in hypercube computers," in G. C. Fox, editor, *The Third Conference on Hypercube Concurrent Computers and Applications, Volume 1*, pages 222–231. ACM Press, 11 West 42nd Street, New York, NY 10036, January 1988.

[lyzenga:88] Lyzenga, G. A., Raefsky, A., and Nour-Omid, B. "Implementing finite element software on hypercube machines," in G. C. Fox, editor, *The Third Conference on Hypercube Concurrent Computers and Applications, Volume 2*, pages 1755–1761. ACM Press, 11 West 42nd Street, New York, NY 10036, January 1988. Caltech Report C3P-594.

[simic:88a] Simic, P. "Statistical mechanics and back propagation learning: Novel approach to learning in feed-forward networks." Technical Report C3P-696, California Institute of Technology, 1988. in preparation.

[simic:88b] Simic, P. "Statistical mechanics and master/slave approach to content addressable memories." Technical Report C3P-697, California Institute of Technology, 1988. in preparation.

[terzopoulos:83] Terzopoulos, D. "Multilevel computational processes for visual surface reconstruction," *Computer Vision Graphics and Image Processing*, 24:52–96, 1983.

[terzopoulos:86] Terzopoulos, D. "Image analysis using multigrid relaxation methods," *IEEE Transactions PAMI*, March 1986.

[williams:87] Williams, R. D. "Dynamical grid optimization for lagrangian hydrodynamics." Technical Report C3P-424, California Institute of Technology, April 1987.

[williams:88a] Williams, R. D. "DIME: A programming environment for unstructured triangular meshes on a distributed-memory parallel processor," in G. C. Fox, editor, *The Third Conference on Hypercube Concurrent Computers and Applications, Volume 2*, pages 1770–1787. ACM Press, 11 West 42nd Street, New York, NY 10036, January 1988. Caltech Report C3P-502.

[williams:88d] Williams, R. "Free-lagrange hydrodynamics with a distributed-memory parallel processor," *Parallel Computing*, 7:439–443, 1988. North-Holland. Caltech Report C3P-424b.