European Regional Science Association
38th European Congress in Vienna, Austria
August 28 – September 1, 1998

A Genetic-Algorithms Based Evolutionary Computational Neural Network for Modelling Spatial Interaction Data

Manfred M. Fischer

Institute for Urban and Regionl Research,
Austrian Academy of Sciences
A-1010 Vienna
Postgasse 7/4
and
Department of Economic and
Social Geography,
University of Economics and
Business Administration
A-1090 Vienna
Augasse 2-6
e-mail: Manfred.Fischer@wu-wien.ac.at

Yee Leung

Department of Geography
and
Center for Environmental Studies,
The Chinese University of Hongkong
Shatin, N.T.
HONGKONG
e-mail: yeeleung@cuhk.edu.hk

Abstract

Building a feedforward computational neural network model (CNN) involves two distinct tasks: determination of the network topology and weight estimation. The specification of a problem adequate network topology is a key issue and the primary focus of this contribution. Up to now, this issue has been either completely neglected in spatial application domains, or tackled by search heuristics (see Fischer and Gopal 1994). With the view of modelling interactions over geographic space, this paper considers this problem as a global optimization problem and proposes a novel approach that embeds backpropagation learning into the evolutionary paradigm of genetic algorithms. This is accomplished by interweaving a genetic search for finding an optimal CNN topology with gradient-based backpropagation learning for determining the network parameters. Thus, the model builder will be relieved of the burden of identifying appropriate CNN-topologies that will allow a problem to be solved with simple, but powerful learning mechanisms, such as backpropagation of gradient descent errors. The approach has been applied to the family of three inputs, single hidden layer, single output feedforward CNN models using interregional telecommunication traffic data for Austria, to illustrate its performance and to evaluate its robustness.

1. Introduction

The recent emergence of computational intelligence technologies such as artificial life, evolutionary computation and neural networks has been accomplished by a virtual explosion of research, spanning a range of disciplines, perhaps wider than any other contemporary intellectual endeavour. Researchers from such diverse fields such as neuroscience, computer science, cognitive science, physics, engineering, statistics, mathematics, computational economics and GeoComputation are daily making substantial contributions to the understanding, development and applications of computational adaptive systems.

With a few exceptions (notably Openshaw 1988, 1993, 1997, Leung 1994, 1997, Fischer 1997, Fischer et al. 1997, Fischer and Gopal 1994, Gopal and Fischer 1996, Openshaw and Openshaw 1997, Nijkamp et al. 1996) geographers and regional scientists have been rather slow in realizing the potential of these novel technologies for spatial modelling. Recently, neural spatial interaction models with three inputs and a single output have been established as a powerful class of universal function approximators for spatial interaction flow data (see Fischer and Gopal 1994).

One of the open issues in neural spatial interaction modelling includes the model choice problem, also termed the problem of determining an appropriate network topology. It consists of optimizing the complexity of the neural network model in order to achieve the best generalization. Considerable insight into this phenomenon can be obtained by introducing the concept of the bias-variance trade-off, in which the generalization error is disaggregated into the sum of the squared bias plus the variance. A model that is too simple, or too inflexibe, will have a large bias, while one that has too much flexibility in relation to the particular data set will have a large variance. The best generalization is obtained when the best compromise between the conflicting requirements of small bias and small variance are achieved. In order to find the optimal balance between the bias and the variance it is necessary to control the effective complexity of the model, complexity measured in terms of the number of adaptive parameters (Bishop 1995).

Various techniques have been developed in the neural network literature to control the effective complexity of neural network models, in most cases as part of the network training process itself. The most widely used approach is to train a set of model candidates and choose that one which gives the best value for a generalization performance criterion. This approach requires significant computational effort and yet it only searches a restricted class of models. An obvious drawback of such an approach is its trial and error nature. An alternative and a more principled approach to the problem utilized by Fischer et al. (1997) is to start with an ‘oversized’ model and gradually remove either parameter weights or complete processing units in order to arrive at a suitable model. This technique is known as *pruning technique*. One difficulty with such a technique is associated with the threshold definitions that are used to decide which adaptive parameters or processing units are important.

Yet another way, to optimize the model complexity for a given training data set is the procedure of *stopped* or *cross-validation* that had been used by Fischer and Gopal (1994). Here, an overparameterized model is trained until the error on further independant data, called validation data set, deteriorates, then training is stopped. This contrasts to the other above approaches since model choice does not require convergence of the training process. The training process is used to perform a directed search of the weight space for a model that does not overfit the data and, thus, demonstrates generalization performance. This approach has its shortcomings too. First, it might be hard in practice to identify when to stop training. Second, the results may depend on the specific training set-validation set pair chosen. Third, the model which has the best performance on the validation set might not be the one with the best performance on the test set.

Though these approaches address the problem of neural network model choice, they investigate only restricted topological subsets rather than the complete class of computational neural network (CNN) architectures. As a consequence, these techniques tend to force a task into an assumed architectural class rather than fitting an appropriate architecture to the task. In order to circumvent this deficiency, we suggest genetic algorithms, a rich class of stochastic global search methods, for determining optimal network topologies. Genetic search on the space of CNN topologies relieves the model builder of the burden of identifying the network structure (topology) that would otherwise have to be done by hand using trial and error. Standard genetic algorithms, with no tricks to speed up convergence, are very robust and effective for global search, but very slow in fine-tuning (i.e. converging) a good solution once a promising region of the search space has been identified (Maniezzo 1994). This motivates one to marry the advantages of genetic evolution and gradient-based (local) learning. Genetic algorithms can be used to provide a model of evolution of the topology of CNNs, and supervised learning may be utilized to provide simple, but powerful learning mechanisms. Backpropagation learning appears to be a natural local search integration for genetic evolution, in the case of CNN optimization.

The remainder of this paper is organized as follows. Section 2 describes the basic features of neural spatial interaction models along with gradient-based backpropagation learning as standard approach to parameter estimation. Section 3 introduces the fundamentals of genetic algorithms and concludes with a brief overview of how they can be applied to network modelling. Section 4 presents the hybrid system, called GENNET (standing for GENetic evolution of computational neural NETworks) that interweaves a genetic search for an appropriate network topology (in the space of CNN topologies) with gradient-based backpropagation learning (in the weight space) for determining the network parameters. Modelling spatial interaction data has special significance in the historical development of mathematical modelling in geography and regional science, the testing ground for new approaches. The testbed for the evaluation uses interregional telecommunication traffic data from Austria because they are known to pose a difficult problem to neural networks using backpropagation learning due to multiple local minima and there is a CNN benchmark available (see Fischer and Gopal 1994, Gopal and Fischer 1996). Section 5 reports on a set of experimental tests carried out to identify an optimal parameter setting and to evaluate the robustness of the approach suggested with respect to its parameters, using a measure which provides an appropriate compromise between network complexity and in-sample - and out-of-sample performances. Section 6 summarizes the results achieved, and outlines directions for future research.

2. Neural Spatial Interaction Models

Neural spatial interaction models are termed *neural* in the sense that they have been inspired by neuroscience. But they are more closely related to conventional spatial interaction models of the gravity type than they are to neurobiological models. They are special cases of general feedforward neural network models. Rigorous mathematical proofs for the universality of such models employing continuous sigmoid type transfer functions (see among others Hornik et al. 1989) establish the three input-single output-single hidden layer neural spatial interaction models developed by Fischer and Gopal (1994) as a powerful class of universal approximators for spatial interaction flow data.

Such models may be viewed as a particular type of an input-output model. Given a three-dimensional input vector **x** that represents measures of origin propulsiveness, destination attractiveness and spatial separation, the neural model produces a one-dimensional output vector **y**, say

$$\mathbf{y} = \Phi(\mathbf{x},\mathbf{w}) = \psi\left(\sum_{j=0}^{J}\beta_j\varphi_j\left(\sum_{n=0}^{3}\alpha_{jn}x_n\right)\right) \qquad (1)$$

representing spatial interaction flows from regions of origin to regions of destination. J denotes the number of hidden units, $\varphi_j(.)$ (j=1, ..., J) and $\psi(.)$ are transfer (activation) functions of, respectively, the j-th hidden and the output unit. The symbol **w** represents a (5J+1)-dimensional vector of all the α- and β-network weights (parameters). x_0 represents a bias signal equal to 1. The transfer functions $\varphi_j(.)$ and $\psi(.)$ are assumed to be differentiable, non-linear; moreover, $\varphi_j(.)$ is generally, but not necessarily assumed to be identical for j=1, ..., J.

Each neural spatial interaction model $\Phi(\mathbf{x},\mathbf{w})$ can be represented in terms of a network diagram (see Fig. 1) such that there is a one-to-one correspondence between components of Φ and the elements of the diagram. Equally, any topology of a three layer network diagram with three inputs and a single output, provided it is feedforward, can be translated into the corresponding neural spatial interaction model. We can, thus, consider model choice in terms of topology selection [i.e., choice of the number of hidden units] and specification of the transfer functions ψ and φ_j (j=1, ..., J).

When approximating the analytically unknown input-output function $\mathbf{F}: \Re^3 \rightarrow \Re$ from available samples (x^k, y^k) with $\mathbf{F}(\mathbf{x}^k)=\mathbf{y}^k$), we have to determine the structure [i.e., the choice of ψ and φ_j with j=1, ..., J, and the network topology] of the spatial interaction model Φ first, and from that finding an optimal set $\mathbf{w}^{opt}$ of adaptive parameters. Obviously, these two processes are intertwined. If a good set of transfer functions can be found, the success of which depends on the particular real world problem, then the task of weight learning [parameter estimation] generally becomes easier to perform.

In all the models under investigation, the hidden unit and output unit transfer functions [φ_j with j=1, ..., J and ψ] are chosen to be identical and the logistic function. This specification of the general model class Φ leads to neural spatial interaction models, say Φ^L, of the following type

$$y = \Phi^L(\mathbf{x}, \mathbf{w}) = \left\{ 1 + \exp\left[-\lambda \sum_{j=0}^{J} \beta_j \left[1 + \exp\left(-\lambda \sum_{n=0}^{3} \alpha_{jn} x_n \right) \right]^{-1} \right] \right\}^{-1} \qquad (2)$$

with values λ close to unity.

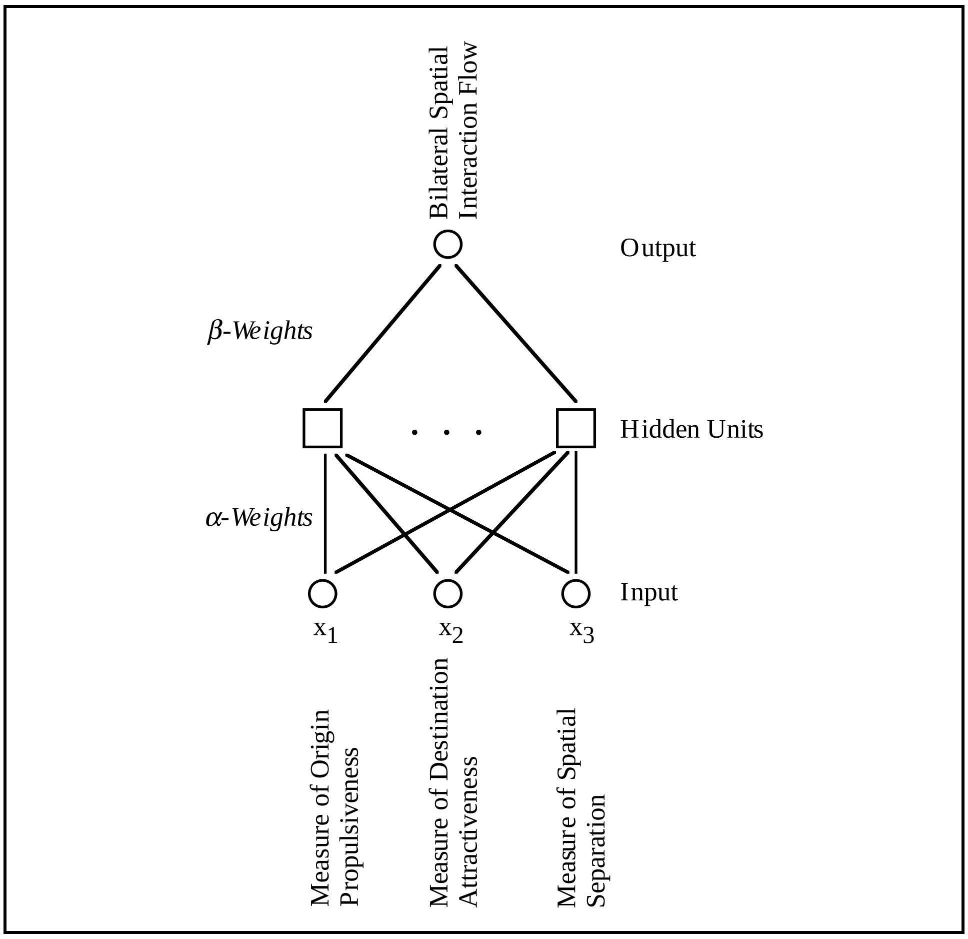
Bilateral Spatial Interaction Flow
Output
β-Weights
Hidden Units
α-Weights
Input
x1
x2
x3
Measure of Origin Propulsiveness
Measure of Destination Attractiveness
Measure of Spatial Separation

Fig. 1 Representation of the general class of neural spatial interaction models defined by equation (1) [biases not shown]

Thus, the problem of determining the model structure is reduced to determine the network topology of the model [i.e., the number J of hidden units]. Hornik et al. (1989) have demonstrated with rigorous mathematical proofs that network output functions such as Φ^L can provide an accurate approximation to any function F likely to be encountered, provided that J is sufficiently large. This universal approximation property establishes the attractivity of the spatial interaction models considered in this contribution.

Without loss of generality, we assume Φ^L to have a fixed topology, i.e. J is predetermined. Then, the role of learning is to find suitable values for network weights $\mathbf{w}$ of this model such that the underlying input-output relationship F: $\Re^3 \rightarrow \Re$ represented by the training set $(\mathbf{x}^k, \mathbf{y}^k)$ k=1, 2, ...; i.e., $\mathsf{F}(\mathbf{x}^k)=\mathbf{y}^k$, is approximated or learned, where k indexes the training instance. $\mathbf{y}^k$ is a 1-dimensional vector representing the desired network output [i.e. the spatial interaction flows] upon presentation of $\mathbf{x}^k$ [i.e. measures of propulsiveness, destination attractiveness and spatial separation]. Since the learning here is supervised (i.e., target outputs $\mathbf{y}^k$ are available), an error (objective, performance) function may be defined to measure the degree of approximation for any given setting of the network's weights. A commonly used, but by no means the only error function is the least squares criterion which is defined for on-line learning as follows

$$\mathrm{E}(\mathbf{w}) = \tfrac{1}{2} \sum_{(\mathbf{X}^k,\, \mathbf{Y}^k)} \left(\mathbf{y}^k - \Phi\left(\mathbf{x}^k,\, \mathbf{w}\right)\right)^2. \qquad (3)$$

Once a suitable error function is formulated, learning can be viewed as an optimization process. That is, the error function serves as a criterion function, and the learning algorithm seeks to minimize the criterion function such as (3) over the space of possible weight settings. Using (3) an optimal parameter set $\mathbf{w}^{opt}$ may be chosen as:

$$\mathbf{w}^{opt}: \qquad \mathrm{E}(\mathbf{w}^{opt}) = \min_{\mathbf{W}} \mathrm{E}(\mathbf{w}). \qquad (4)$$

The most prominent learning algorithm which has been proposed in the neural network literature to solve this minimization problem is backpropagation (BP) learning (Rumelhart, Hinton and Williams 1986) combined with the gradient descent technique which allows for efficient updating of the parameters due to the feedforward architecture of the spatial interaction models.

In its standard version backpropagation learning starts with an initial set of random weights $\mathbf{w}_0$ and then updates them by

$$\mathbf{w}_\tau = \mathbf{w}_{\tau\text{-}1} + \eta\, \nabla\Phi(\mathbf{x}^k, \mathbf{w}_{\tau\text{-}1})\, (\mathbf{y}^k - \Phi(\mathbf{x}^k, \mathbf{w}_{\tau\text{-}1})) \qquad k=1, 2, ..., K \qquad (5)$$

where $\mathbf{w}$ is the (5J+1)-dimensional vector of network weights to be learned; its current estimate at time τ-1 is denoted by $\mathbf{w}_{\tau\text{-}1}$; ($\mathbf{x}^k$, $\mathbf{y}^k$) is the training pattern presented at time k; Φ is the network function; η is a fixed step size (the so-called learning rate), $\nabla\Phi$ is the gradient (the vector containing the first-order partial derivatives) of Φ with respect to the parameters $\mathbf{w}$. Note that parameters are adjusted in response to errors in hitting the target, $\mathbf{y}^k$- Φ($\mathbf{x}^k$, $\mathbf{w}_{\tau\text{-}1}$). The performance of backpropagation learning can be greatly influenced by the choice of η. Note that (5) is the parameter update equation of the on-line, rather than the batch version of the backpropagation learning algorithm. For very small η (i.e. approaching zero) on-line backpropagation learning approaches batch backpropagation (Finnoff 1993). But there is a non-negligible stochastic element [i.e. ($\mathbf{x}^k$, $\mathbf{y}^k$) are drawn at random] in the training process that gives on-line backpropagation a quasi-annealing character in which the cumulative gradient is continuously perturbed, allowing the search to escape local minima with small and shallow basins of attraction (Hassoun 1995). Although many modifications of this procedure [notably the introduction of a momentum term, $\mu\ \Delta\mathbf{w}_{\tau\text{-}1}$, into the weight update equation and the use of a variable step size, denoted $\eta_{\tau\text{-}1}$] and alternative optimization procedures have been suggested over the past few years, experience shows that surprising good network performance can often be achieved with this on-line (local) learning algorithm in real-world applications [see, e.g., Fischer et al. 1997 for an epoch-based version].

As the optimum network size and topology [i.e. the number of hidden layers and hidden units, connectivity] are usually unknown, the search of this optimum requires a lot of networks to be trained on a trial and error basis. Moreover, there is no guarantee that the network obtained is globally optimal. In this contribution, we view this issue as a global optimization problem and, therefore, suggest the application of genetic algorithms which provides multi-point global optimal search for the network topology.

3. Basics of the Canonical Genetic Algorithm

Genetic algorithms (GAs) are revealing to be a very rich class of stochastic search algorithms inspired by evolution. These techniques are population oriented and use selection and recombination operators to generate new sample points in a search space. This is in contrast to standard programming procedures that usually follow just one trajectory (deterministic or stochastic), perhaps repeated many times until a satisfactory solution is reached. In the GA approach, multiple stochastic solution trajectories proceed simultaneously, permitting various interactions among them towards one or more regions of the search space. Compared with single-trajectory methods, such as simulated annealing, a GA is intrinsically parallel and global. Local ‘fitness’ information from different members is mixed through various genetic operators, especially the crossover mechanism, and probabilistic soft decisions are made concerning removal and reproduction of existing members. In addition, GAs require only simple computations, such as additions, random number generations, and logical comparisons, with the only major burden that a large number of fitness function evaluations have to be performed (Qi and Palmieri 1994). This section will review the fundamentals of the canonical genetic algorithm as introduced by Holland (1975), and then show how genetic algorithms can be used as means to perform the task of model choice [topology optimization] in the spatial interaction arena.

In its simplest form, the canonical genetic algorithm is used to tackle static discrete optimization problems of the following form:

$$\max \{\mathsf{f}(\mathbf{s}) \mid \mathbf{s} \in \Omega\} \tag{6}$$

assuming that $0<\mathsf{f}(\mathbf{s})<\infty$ for all $\mathbf{s}\in\Omega=\{0, 1\}^d$ and $\mathsf{f}(\mathbf{s}) \neq$ const. In this case, the objective function is called the *fitness function* $\mathsf{f}: \Omega\rightarrow\Re$, and the d-dimensional binary vectors in Ω are called *strings* [sometimes also genotypes or chromosomes]. The size of the search space Ω is 2^d and forms a hypercube. The GA samples the corners of this d-dimensional hypercube. The P-tuple of individual strings $(\mathbf{s}_1, ..., \mathbf{s}_P)$ is said to be a *population* S. Each individual $\mathbf{s}_k \in$ S represents a feasible solution of problem (6) and its objective function value $\mathbf{f}(\mathbf{s}_k)$ is said to be its *fitness* which is to be maximized. $\mathbf{f}$ is called the fitness function. If the problem is to minimize a given objective function g

$$\min \{g(\mathbf{x}) \mid \mathbf{x} \in \Sigma\} \tag{7}$$

assuming that $0<g(\mathbf{x})<\infty$ for all $\mathbf{x} \in \Sigma \subset \Re^d$ and $g(\mathbf{x}) \neq$ const, then it is necessary first, to map the 'real' search space Σ into the representation space Ω of binary [fixed-length] strings, and, second, to transform the objective function $g(\mathbf{x})$ into an appropriate fitness function $\mathbf{f}(\mathbf{s})$ such that the maximizers of $\mathbf{f}(\mathbf{s})$ correspond to the minimizers of $\mathbf{x}$. This situation is schematically illustrated in Fig. 2.

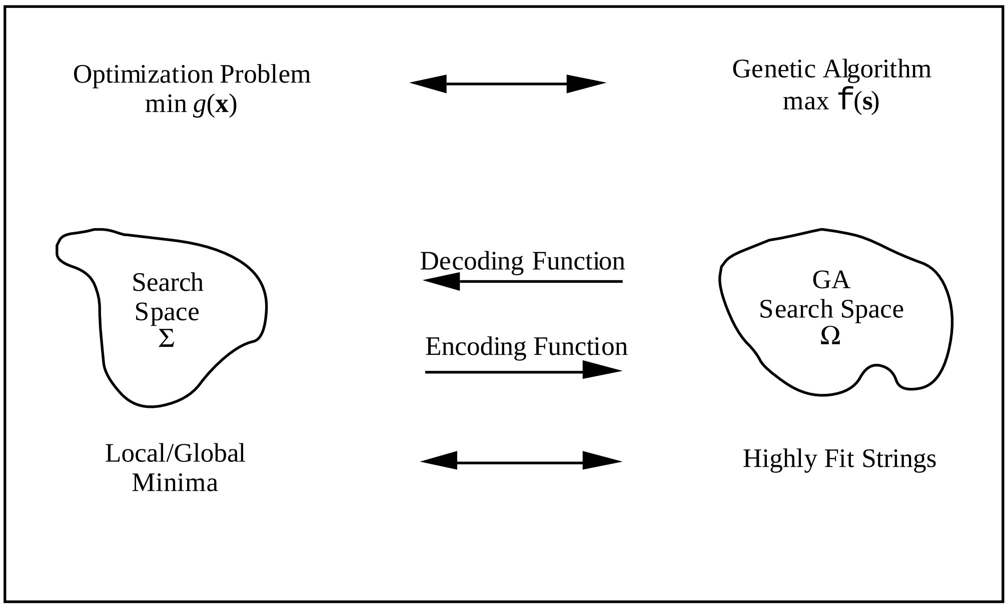

Fig. 2 The dual representation scheme used in the GA-approach [see Hassoun 1995]

The standard genetic algorithm can be sketched as follows:

Step 1: Randomly generate an initial population of, say, P binary strings of length d: $S(0)=\{\mathbf{s}_1, ..., \mathbf{s}_P\} \subset \Omega$.

Step 2: Compute the fitness score $\mathbf{f}(\mathbf{s}_k)$ of each individual string $\mathbf{s}_k$ of the current population S(t).

Step 3: Generate an intermediate population [termed mating pool] by applying the *selection* operator.

Step 4: Generate S(t+1) by applying recombination operators (*crossover* and *mutation*) to the intermediate population.

Step 5: t:=t+1 and continue with *Step 2* until some stopping criterion applies [in this case designate the best-so-far individual as the result of the GA].

The first step generates an initial population S(0), i.e. $S(0)=\{\mathbf{s}_1, ..., \mathbf{s}_P\}\subset\Omega$. In the canonical GA each member of S(0) is a binary string of length d that corresponds to the problem coding. S(0) is usually generated randomly, because it is not known a priori where the globally optimal strings in Ω are likely to be found. From this initial population, subsequent populations S(1), ..., S(t), ... will be computed by employing the three genetic operators of selection (reproduction), crossover and mutation.

After calculating the relative fitness for all the strings in the current population S(t) (*Step 2*), selection is carried out. In the canonical GA the *roulette wheel selection* (Goldberg 1989) technique is used for constructing the intermediate population (*Step 3*), i.e. a proportional selection technique, where the intermediate population is determined by P independent random experiments. The probability that individual $\mathbf{s}_k$ is selected from tuple $(\mathbf{s}_1, ..., \mathbf{s}_P)$ to be member of the intermediate population at each experiment is given by

$$p_r\,(\mathbf{s}_k \text{ is selected}) = \left[f(\mathbf{s}_k) / \sum_{k=1}^{P} f(\mathbf{s}_k) \right] > 0 \qquad (8)$$

That is, strings in the current population are copied (i.e. duplicated) and placed in the intermediate population proportional to their fitness relative to other individuals in the population.

After selection has been carried out the construction of the intermediate population is complete. Then crossover and mutation are applied to the intermediate population to create the next population S(t+1) (*Step 4*). Crossover and mutation provide a means of generating new sample points in Ω while partially preserving distribution of strings across hyperplanes which is observed in the intermediate population. *Crossover* is a recombination mechanism to explore new regions in Ω. The crossover operator is applied with some probability $p_c \in [0, 1]$. To apply the one-point crossover operator, for example, the individuals of the intermediate population are randomly paired. Each pair (*parents*) is then combined, choosing one point in accordance with a uniformly distributed probability over the length of the individual strings and cutting them in two parts accordingly. The two new strings, called *offspring*, are formed by the juxtaposition of the first part of one parent and the last part of the other parent. For example, Fig. 3 illustrates a crossover for two 7-bit strings. In this case, the crossing site is 5, so the bits from the two strings are swapped after the fifth bit. It should be noted that we can also perform multi-point crossover (Spears and De Jong 1991) and uniform crossover (Syswerda 1989) for evolution.

(a)	(b)	(c)
0 1 0 1 1 0 0	**0 1 0 1 1**| 0 0	0 1 0 1 1 0 1
1 1 0 0 1 0 1	1 1 0 0 1| **0 1**	1 1 0 0 1 0 0

Fig. 3 An example of a one-point crossover for 7-bit strings: (a) two strings selected for crossover, (b) a crossover site is selected at random, (c) the two strings are swapped after the 5th bit

Finally, after crossover, the *mutation* operator is applied with uniform probability p_m. The operation is a stochastic bit-wise complementation and serves the important, but secondary role of ensuring that the entire representation space Ω remains accessible. Mutation may be applied to offspring produced by crossover or, as an independent operator, at random to any individual in the intermediate population. It operates independently on each individual by probabilistically perturbing each bit string. The event that the i-th bit of the k-th individual is flipped from 0 to 1 or from 1 to 0 is stochastically independent and occurs with probibility $p_m \in [0, 1]$. As an example, assume p_m=0.1, and the string 0101101 is to undergo mutation. The easiest way to determine which bits, if any, to flip is to select a uniform random number $r \in [0, 1]$ for each bit in the string. If $r \leq p_m = 0.1$, then the bit is flipped, otherwise not. Suppose that the random numbers (0.30, 0.91, 0.05, 0.54, 0.48, 0.89, 0.17) were generated for the string in question, then the third bit has to be flipped and the resulting string is 01**1**1101. After mutation, the candidate strings are copied into the new population S(t+1) of strings, and the whole process is repeated by decoding each individual into a form appropriate for evaluation, calculating its fitness, using a roulette wheel method of selection, and applying the operators of crossover and mutation.

It is important to note that as the average evaluation of the strings in the population increases, the variance in fitness decreases in the population. After some generations there may be little difference between the best and worst individual in the population, and the selective pressure based on fitness is correspondingly reduced. This problem can be partially addressed by using some form of fitness scaling (Goldberg 1989). In the simplest case, one can subtract the evaluation of the worst string in the population from the evaluations of all strings in the population. One can now calculate the average stringe evaluation as well as fitness values using this adjusted evaluation. This will increase the resulting selective pressure.

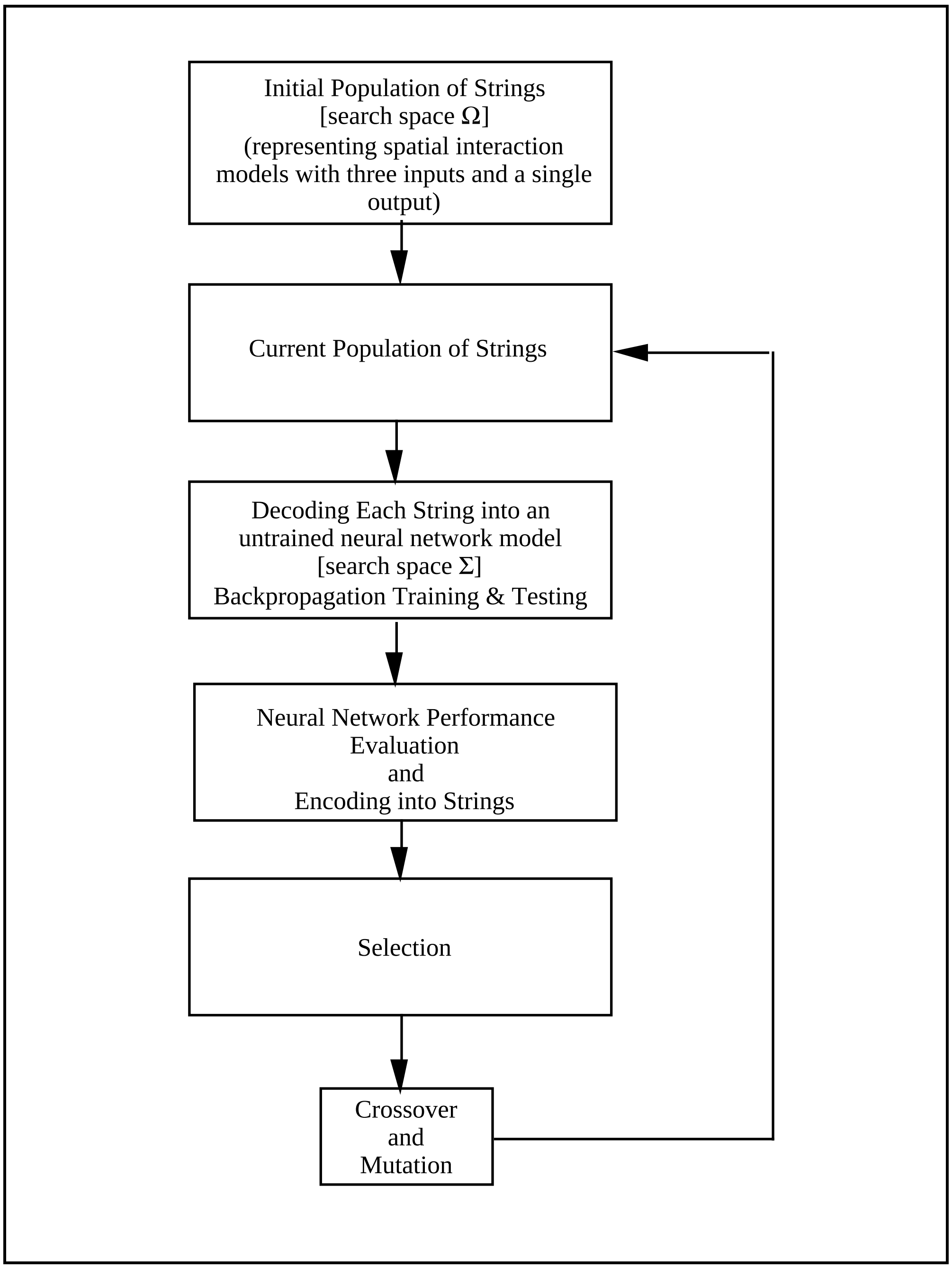
Initial Population of Strings
[search space Ω]
(representing spatial interaction models with three inputs and a single output)
Current Population of Strings
Decoding Each String into an untrained neural network model
[search space Σ]
Backpropagation Training & Testing
Neural Network Performance Evaluation and Encoding into Strings
Selection
Crossover and Mutation

Fig. 4 The principle structure of a genetic approach for the neural network topology optimization problem

The general idea to use the GA to select a topology (pattern of connections) for the neural spatial interaction models is illustrated in Fig. 4. This approach is inherently computationally demanding because the complete conventional training phase (itself computationally intensive) is required to simply evaluate the fitness of a string [i.e. a neural network topology]. But the approach remains reasonably attractive despite this because of the scarcity of fully automated alternative procedures for optimally selecting the network topology. It is especially useful and effective when one needs to find the optimal topology of a highly complex neural network which cannot be assumed by the trial-and-error or heuristic procedures.

While the procedure outlined in Fig. 4 is quite straightforward, the problem of combining GA and neural spatial interaction modelling, however, lies in the definitions of an invertible transformation to encode/decode the original search space Σ into some GA-space Ω, and of the fitness evaluation function. These are the two main components of most genetic algorithms that are problem dependent.

In principle, three major types of direct encoding strategies may be used: weight-based, node-based, layer-based and pathway-based strategies [opposed to indirect encoding where rules or alikes are encoded that carry information on how the CNN has to be constructed]. Node-based strategies do not encode the network weights, but node information such as the number of nodes, connectivity and placement information of the nodes, and the type of transfer functions are encoded. In layer-based encoding schemes the layer size and information about output and input connections are stored inter alia, while in pathway-based encoding, pathways through the network are encoded but not the connections, nodes or layers. It is important to note that the search space Ω [space of strings or representation space] is enormously enlarged if the genetic representation of CNNs distinguishes between networks which differ only by the labelling of hidden units. This problem is known as permutational redundancy associated with the arbitrariness of labels of topologically equivalent hidden nodes, and tends to make genetic navigation very difficult since GAs are sensitive to the potential for redundant representations (Radcliff 1991). Aside from the coding issue, the definition of the fitness function has to be given as part of the problem definition. The fitness function may be defined as a combined measure (=overall performance) which may take into account learning speed, accuracy in terms of out-of-sample (testing) performance and factors such as the size and complexity of the model.

The process of training individual neural network models, measuring their fitness, and applying genetic operators to produce a new population of neural network models is repeated over many generations. Each generation should tend to contain more of the features which were found useful in the previous generation, and an improvement in overall performance can be realized over the previous generation.

The next section serves to discuss the major issues involved in using genetic algorithms for neural network topology optimization. These include the representation of the strings that specifies both the structure and the estimation procedure, the choice of the underlying space Σ of neural network topologies for exploration, adaptations of the basic genetic operators used to construct meaningful neural spatial interaction structures, and the form of the evaluation function which determines the fitness of a neural network.

4. GENNET: A System for Structure Optimization and Weight Determination

This section describes the evolutionary algorithm on which the GENNET system is based (Leung et al., 1995). The system essentially consists of two major modules: a Genetic Algorithm Engine used for designing the topology [structure optimization] and a Network Simulator for gradient-based backpropagation learning and testing. Both modules are interwoven via a Network Decoder and a Network Fitness Evaluator (see Fig. 5). Basically, the Genetic Algorithm Engine encodes neural network topologies as strings and evolves them through genetic operators. The evolved string is then decoded into a neural network by the Network Decoder (generator) and is then fed to the neural network engine for training. Based on the given training patterns, the engine trains the given neural networks by *error backpropagation* of gradient descents, and the resulting networks are then tested with the given testing patterns. Various statistics such as the size of the network, learning speed, in-sample and out-of-sample performance are recorded and passed back to the genetic algorithm engine for fitness evaluation. Networks with high fitness are selected and further processed by various genetic operators. The whole process is repeated until a network with fitness value higher than the specified requirement is found.

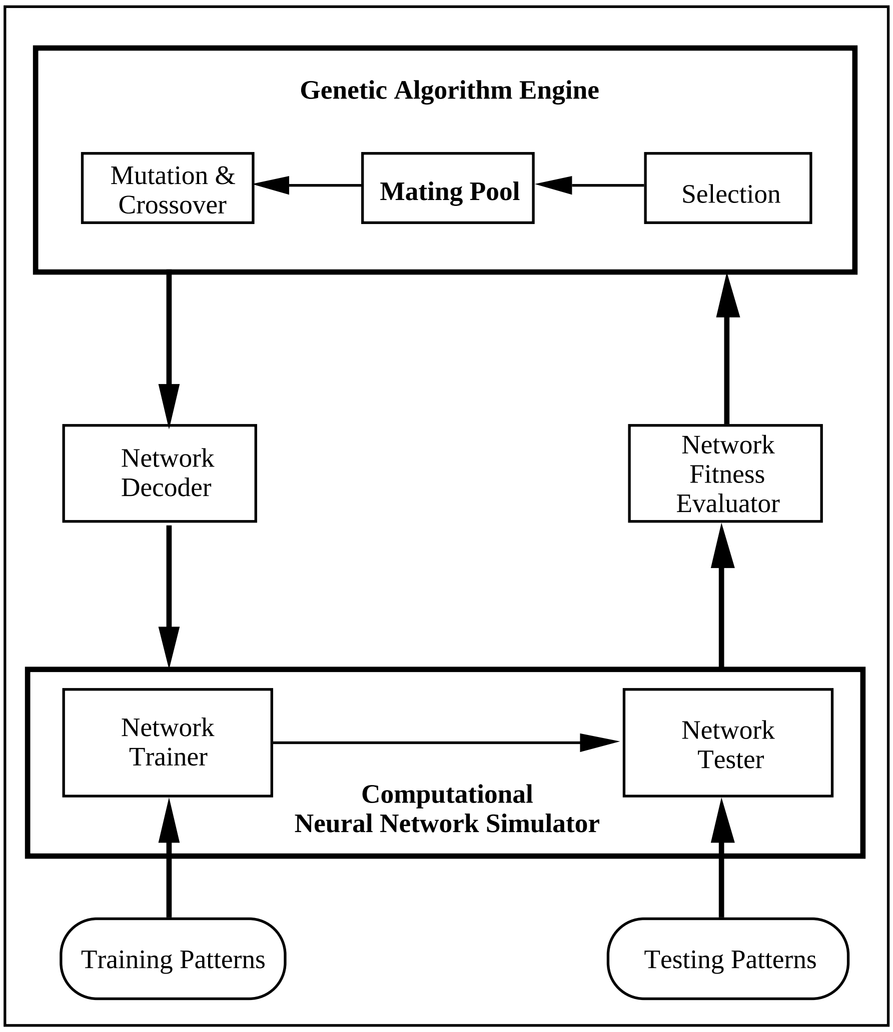

Fig. 5 GENNET: A hybrid approach for optimizing the structure and the weights of multilayer computational neural networks

In the sequel, we detail the structure and the mechanism of the GA-based system. The utilization of genetic algorithms in any specific application entails three related activities: *first*, the definition of an objective function which indicates the fitness of any potential solution, *second*, the definition of the encoding structure, and *third*, the definition of the genetic operators to be used.

Fitness Function: In GENNET, optimal network design corresponds to the global minimum of a fitness function. The fitness function is defined to depend on three factors. The main factor used for the fitness function is the network's ability to generalize unseen patterns, i.e. the generalization or out-of-sample performance measured in terms of the average relative variance. For this purpose, we used a validation test set in comparison to the training set. To create a selective pressure that favours smaller networks, a parameter reflecting the network complexity (in terms of the number of hidden layers and the number of their nodes) is also included in the fitness function. Finally, the number of cycles required to train the network [i.e. the speed of convergence] in comparison to the maximum number of cycles utilized so far is taken into consideration, even though this parameter is not directly associated with neural network topology. This leads to the following fitness function

$$\text{Fitness} = 1 - r_1 \text{ TestF} - r_2 \text{ NCF} - r_3 \text{ TrainF} \tag{9}$$

where TestF represents the generalization performance, NCF the network complexity and TrainF the speed of convergence. r_1, r_2 and r_3 are scaling factors between zero and one. They may be interpreted as penalty coefficients and have to be carefully adjusted.

Although the fitness function in (9) is relatively simple, its values depend on several ‘hidden’ factors which are not directly associated with network topology. For example, evaluating a given network simply on learning results and validation tests has several drawbacks. For example, using a small number of samples in both phases speeds up the optimization process but may result in removing connections too forcefully since the limited number of measurements might not provide enough evidence to justify the importance of some links. On the other hand, using a larger quantity of data to evaluate the CNN may imply that bigger networks could be trained more precisely than smaller ones and, thus, the implicit pruning process would be reluctant to remove links.

Encoding Scheme: Table 1 illustrates how a string is built. The string representation has several desirable properties. Since all bits are treated uniformly, all genetic operators designed for binary strings can then be applied without modification. The string can be used to encode any initial network architecture. Neural spatial interaction models of class (2) are encoded as a 68-bit string. The following string

```
          1         2         3         4         5         6
12345678901234567890123456789012345678901234567890123456789012345678
10111100000000000000000100100111010101010001000000000001000000000001
```

represents a neural spatial interaction model with a single hidden layer of 30 units [i.e. total number of hidden layers: 1, number of nodes in the hidden layer: 30, connectivity: 1]. Its detailed decoding is as follows:

bit	1	2-7	8	9-14	15	16-21	22-24	25-27	28-30	31-40	41-68
binary	1	011110	0	000000	0	0000000	001	001	001	1101010101	0001000000000000010000000000001
decimal	1	30	0	0	0	0	1	1	1	853	16781313

upper weight limit = $\frac{1}{7} \cdot$ (range of weight limit)

lower weight limit = $\frac{1}{7} \cdot$ (range of weight limit)

momentum limit = $\frac{1}{7} \cdot$ (range of momentum limit)

probability of connection of each layer = $\frac{852}{2^{10-1}} = \frac{853}{1023} = 0.8338$

sigmoid variable = $16781313/2^{28-1})$

= 16781313/268435455*(range of λ)

Table 1 Genetic-algorithm encoding of a multilayer neural network spatial interaction model

Bits	**Meaning**
1	Present flag* of the first hidden layer
2-7	Density parameter: number of nodes in the first hidden layer
8	Present flag* of the second hidden layer
9-14	Density parameter: number of nodes in the second hidden layer
15	Present flag* of the third hidden layer
16-21	Density parameter: number of nodes in the third hidden layer
22-24	determine upper connection weight limit (has 8 discrete levels)
25-27	determine lower connection weight limit (has 8 discrete levels)
28-30	momentum limit (has 8 discrete levels)
31-40	probability of connection of each layer
41-68	sigmoid slope (λ) of the logistic function

*) The flag is used to indicate whether the hidden layer exists or not; 1: present and 0: absent

Operators: *The Genetic Algorithm Engine employs the three genetic operators as described in section 3.* For selection, the standard roulette wheel selection operator, with Monte Carlo selection with probabilities based on fitness level is used. Strings with higher fitness values have a higher chance to be selected for reproduction. For crossover, a random crossover point which is a number between 1 and 67 [string length minus 1] is selected as the position for crossover. For mutation, the standard operator which negates a bit with probability p_m is utilized. Having initialized the evolutionary process with a randomly generated population of strings, the genetic algorithm engine then applies the three genetic operators for reproduction.

All strings generated by the genetic algorithm engine are decoded as neural networks by the **Network Decoder** (generator). At the initialization of the system, the network decoder is initialized with the following information: maximum number of hidden layers [in the current study: up to one hidden layer], minimum and maximum number of nodes in each layer, lower and upper bounds of the weights. Based on the information, the network decoder converts the string into a multilayer neural network. Each weight of the connections is initialized with a random value between the lower and upper bounds of the weights. The decoded network is fed to the network trainer for backpropagation training.

The **Network Trainer** is responsible for the training of the neural network by backpropagating gradient descent errors and using the training set, provided until any one of the following conditions is fulfilled: number of maximum training cycles is reached, mean squared error [i.e. the error function to be minimized during training] is smaller than the desired value, or the mean error improvement with a cycle period is converged. When the training is completed, the network trainer outputs a fitness value: Training Fitness (TrainF), a value within (0, 1) which corresponds to how well the neural network is trained [measured in terms of the error function in (3)]. We assume that the smaller the mean squared error, the greater is the TrainF.

The **Network Tester** is responsible for the testing of the trained network using the testing validation set provided. When the testing is finished, the out-of-sample performance or Testing Fitness (TestF), which is a value within (0, 1), is obtained. TestF, which is monitored by the Network Tester, is measured in terms of the average relative performance (ARV) defined by Fischer and Gopal (1994) as

$$\text{ARV} = \frac{\sum_l \left\| \mathbf{y}^l - \Phi^L(\mathbf{x}^l, \mathbf{w}) \right\|^2}{\sum_l \left\| \mathbf{y}^l - \bar{\mathbf{y}} \right\|^2} \tag{10}$$

where ($\mathbf{x}^l$, $\mathbf{y}^l$) denotes the l-th testing pattern, $\bar{\mathbf{y}}$ the average of the target values $\mathbf{y}^l$ in the testing data set, and Φ^L ($\mathbf{x}^l$, $\mathbf{w}$) the actual model output. This performance measure provides a normalized mean squared error metric for comparing the generalization performance of different CNN models.

The tested network is then evaluated by the **Network Fitness Evaluator** that is responsible for the evaluation of the overall fitness of the neural network. Network Fitness (NF) is based on the Network Complexity Fitness (NCF) [defined as size of the network/maximum possible size], Training Fitness (TrainF), and Testing Fitness (TestF), and is defined as

$$\text{NF} = r_{\text{TestF}}\ \text{TestF} + r_{\text{NCF}}\ \text{NCF} + r_{\text{TrainF}}\ \text{TrainF} \tag{11}$$

where r_{TestF}, r_{NCF}, and r_{TrainF} are constants which reflect the degrees of significance of NCF, TrainF, and TestF respectively. Adjusting the value of r_{TestF}, r_{NCF},, and r_{TrainF} is crucial because the system uses these values as a basis to evolve neural networks of various topologies.

The trained networks are then fed to the GA engine for evolution, especially on the adjustment of the various weights in NF, momentum, probability of connection, and the parameter λ of the logistic function. It should be noted that the relationship between fitness in (9) and NF in (11) of strings [i.e. between the GA-search space Ω and the search space Σ of CNN topologies is given by

$$\text{fitness} = 1 - \text{NF}. \qquad (12)$$

5. Experiments and Performance Tests Using Interregional Telecommunication Traffic Data

Fischer and Gopal (1994) have demonstrated the feasibility of the class of neural spatial interaction models Φ^L to model interregional telecommunication traffic in Austria with noisy real-world data of limited record length. Even though the model identified in this study well outperformed current best practice in form of the classical regression approach of the gravity type, the model selection approach utilized, due to its trial-and-error heuristics, might have considered only restricted subsets of the whole space of CNN topologies. Thus, it is obvious to test and evaluate the suggested approach in this spatial interaction context. To facilitate comparison with the previous work, we considered the same class of neural interaction models with three inputs, a single hidden layer, and a single output unit that represents the intensity of telecommunication flows from one origin region to a destination region. The input units represent the three independent variables of the classical gravity model [i.e. the potential pool of telecommunication activities in the origin region, the potential draw of telecommunication activities in the destination region, and a factor representing the inhibiting effect of geographic separation from the origin to the destination region]. The problem is to identify a model specification with an appropriate complexity in terms of the hidden units to show good generalization [i.e. out-of-sample] performance.

The Data: From three Austrian data sources - a (32, 32)-interregional telecommunication flow matrix, a (32, 32)-distance matrix, and gross regional products for the 32 telecommunication regions - a set of 992 4-tuples $(x_1, x_2, x_3, \mathbf{y})$ was constructed, where the first three components represent the input vector $\mathbf{x}:=(x_1, x_2, x_3)$ and the last component the target output of the CNN model, i.e. the telecommunication intensity from one region of origin to another region of destination. Input and target output signals were preprocessed to logarithmically transformed data scaled into [0, 1]. The telecommunication data stem from network measurements of carried telecommunication traffic in Austria 1991, in terms of erlang, which is defined as the number of phone calls (including facsimile transmissions) multiplied by the average length of the call (transfer) divided by the duration of measurement (for more details see Fischer and Gopal 1994). This data set was randomly divided into two separate subsets: about two thirds of the data were used for parameter estimation only, and one third as validation test set. There was no overlapping of the two sets of data.

Results: A first set of tests was run in order to identify an optimal parameter setting. The genetic algorithm has 4 parameters and there is no way to a priori identify useful combinations of values. The parameters are: P, the size of the population; p_c, crossover probability; p_m, mutation probability; and λ, slope of the logistic transfer function (see equation (2)). In order to define good settings, extensive computational tests with different combinations of values have been performed. All tests were made by letting the algorithm run five times at each setting for 500 function evaluations. The values used for each parameter were: $P \in \{20, 40, 100, 200\}$; $p_c = \{0.5, 0.6, 0.7, 0.8, 0.9\}$; $p_m = \{0.001, 0.01, 0.01, 0.1, 0.2\}$; $\lambda = \{0.1, 0.5, 1, 2, 10\}$. The best settings obtained are summarized in Table 2. Some considerations are in order. *First*, larger populations provide better results, because a large population is more likely to contain representatives from a larger number of hyperplanes. Thus, the GAs can perform a more informed search. But the computational costs of evolving large populations does not seem to be rewarded by comparable improvement of the solutions obtained by, for P>40. *Second*, there is evidence that, within the representation used, the search process benefits from the recombination. Crossover probability equals to the commonly adopted values suggested, for example, by De Jong (1975), i.e. p_c=0.6. If the crossover probabilities are too high, high performance structures are discarded faster than selection can produce improvements. If the probability is too low, the search may stagnate due to the lower exploration rate. *Third*, mutation is a secondary search operator that increases the variability of the population. The mutation probability is small (p_m=0.001), suggesting a larger disruptive impact of the mutations it controls. *Fourth*, the sigmoid slope is a parameter that dramatically affects the results. The best slope value, i.e. λ=2, corresponds to a steeper sigmoid. The experiments also suggest that in smaller populations such as P=40 structures, good performance is associated with either a higher crossover probability combined with a low mutation probability or a lower crossover combined with a higher mutation probability.

Table 2 Best parameter settings

Parameter	Values
P	40
p_c	0.6
p_m	0.001
λ	2
η [learning rate]	0.7
μ [momentum]	0.9

Table 3 reports the generalization [out-of-sample] performance [in terms of ARV] along with the network complexity [j=26 and J=36] of the CNNs evolved over the five trials. Again some considerations are worth making. *First*, compared with the results obtained in Fischer and Gopal (1994) the solutions show a higher variance over the five runs. Sometimes, it moves steadily (though slowly) towards the optimum, sometimes it never greatly modifies the initial fitness, which is close to the half of the optimum. *Second*, the average performance in terms of ARV obtained over the five runs utilizing the validation test patterns is 0.3948 in the case of the less complex network model (J=26) and 0.3879 in the case of the more complex model (J=30). *Third*, for comparative purposes it is worthwhile to mention that the cross-validation approach utilized in Fischer and Gopal (1994) did result in a model with J=30 hidden units and an ARV value of 0.4065 on the test rather than on the validation set averaged over five trials differing in initial conditions as well as in the random sequence of the input signals. But it should be noted that the ARV values are not comparable in a strict sense, because Fischer and Gopal (1994) utilize an additional set of test data, independent from both the training and the validation test sets. The test data set is reserved for the training process, the validation test set for optimizing model complexity [i.e. model choice] and the test set for the evaluation.

Table 3 Performance of the GA solutions [measured in terms of ARV] using the validation test set

	Neural Spatial Interaction Model with J=26 Average Relative Variances	J=30 Average Relative Variances
Run 1	0.588735	0.554846
Run 2	0.306277	0.295533
Run 3	0.294214	0.275657
Run 4	0.417048	0.421113
Run 5	0.367715	0.392486
Average	0.394798	0.387927
Deviation Standard	0.119157	0.111943

The best network model should be a compromise between complexity and performance. In the suggested GA model choice approach, the balance between complexity and performance can be regulated at the level of the fitness function. This is an attractive feature. But care has to be exercised when setting the r-values since the network performance depends also on other 'hidden' aspects of the optimization problem, such as the number of training and validation points in addition to the overall training strategy.

6. Conclusions and Outlook

This paper presents an evolutionary computational approach, called GENNET, suitable for maximizing complex functions, such as those that assign a fitness to a multilayer computational neural network on the basis of its topology, its training performance and convergence speed. Search effectiveness is achieved through a direct encoding procedure that allows CNN topologies to be represented by 68-bits strings, and an algorithm that automatically identifies minimal search space containing reachable solutions at evolution time. The genetic algorithm permits search over the whole search space of CNN topologies, thus, providing the possibility of escaping from local minima. Local search is performed by applying backpropagation learning to the individuals of the GA-population. Interregional telecommunication traffic data from Austria (Fischer and Gopal 1994) serve as a benchmark for evaluating and illustrating the performance of the hybrid aproach where the objective is to generate a three inputs, single output, and single hidden layer computational neural network with logistic hidden units.

As was shown in the simulation studies, GENNET successfully generates an optimal topology for the neural network spatial interaction model without manual intervention required by other heuristic approaches to the model choice problem. Though the computational costs tend to be higher, it is, in fact cost effective if time spent on human supervision and intervention in the other approaches is taken into account. The benefit will be even more apparent when complex network topologies more complex than that in this study have to be evolved.

We believe that GENNET may be made substantially more efficient, for example, by using floating-point rather than binary representations (see Michalewicz 1992). But there is little doubt that it will remain slower in any serial implementation. The greatest potential for the application of this and other types of evolutionary optimization to real world problems will come from their implementation on parallel machines, because evolution is an inherently parallel process. While the evolutionary process itself is slow, the CNN it generates is evolved to be computationally efficient in the sense of producing the best approximation using the fewest computational units, provided the system parameters used in the experiments are optimal for the given problem.

Finally, we should mention that there are several other population based algorithms that are either spinoffs of the canonical genetic algorithm, or independently developed. Evolution strategies and evolutionary programming, for example, are two computational paradigms that use a population based search. Evolutionary programming (see Fogel 1995), in particular, appears to be an attractive alternative to genetic algorithms since it provides the possibility to manipulate CNNs directly and, thus, obviates the need for a dual representation. We leave comparisons between evolutionary programming and genetic algorithms on the problem addressed in this paper for future research.

Acknowledgement: The authors would like to thank K.S. Leung, W. Ng, M.K. Lau and S.K. Cheung for their assistance in implementation; and the Hong Kong Research Grant Council for the support through the earmarked grant CUHK 321/95H.

References

Bishop C M (1995) *Neural networks for pattern recognition*. Clarendon Press, Oxford

Caudele T P, Dolan C P (1989) Parametric connectivity: Training of constrained networks using genetic algorithms In: Schaffer J D (ed) *Proceedings of the Third International Conference on Genetic Algorithms and their Applications*, 370-4. Morgan Kaufmann, San Mateo (CA)

De Jong K A (1975) Analysis of the behavior of a class of genetic adaptive systems. Ph.D. dissertation, University of Michigan

Finnoff W (1993) Diffusion approximations for the constant learning rate backpropagation algorithm and resistance to local minima In: Hanson S J, Cowan J D, Giles C L (eds) *Advances in Neural Information Processing Systems V*, 459-66 Morgan Kaufmann, San Mateo (CA)

Fischer M M (1997) Computational neural networks. A new paradigm for spatial analysis *Enivronment and Planning A* [in press]

Fischer M M, Gopal S (1994) Artificial neural networks A new approach to modelling interregional telecommunication flow *Journal of Regional Science* 34(4), 503-27

Fischer M M, Gopal S, Staufer P, Steinnocher K (1997) Evaluation of neural pattern classifiers for a remote sensing application *Geographical Systems* 4 [in press]

Fogel D B (1994) An introduction to simulated evolutionary optimization *IEEE Transaction on Neural Networks* 5(1), 3-14

Fogel D B (1995) *Evolutionary Computation: Toward a New Philosophy of Machine Intelligence* IEEE Press, Piscataway (NJ)

Goldberg D (1989) *Genetic Algorithms* Addison-Wesley, Reading (MA)

Gopal S, Fischer M M (1996) Learning in single hidden-layer feedforward network models *Geographical Analysis* 28(1), 38-55

Grefenstette, J J (1986) Optimization of control parameters for genetic algorithms *IEEE Transactions on Systems, Man and Cybernetics* SMC 16, 122-8

Hassoun M H (1995) *Fundamentals of Artificial Neural Networks* The MIT Press, Cambridge, MA and London (England)

Hinton G E (1990) Connectionist learning procedures In: Kodratoff Y, Michalski, R (eds) *Machine Learning III*, 555-610 Morgan Kaufmann, San Mateo (CA)

Holland J H (1975) *Adaptation in Natural and Artificial Systems* The University of Michigan Press, Ann Arbor (Michigan)

Hornik K M, Stinchcombe M, White H (1989) Multilayer feedforward networks are universal approximators *Neural Networks* 2, 359-66

Koza J R (1993) *Genetic Programming* The MIT Press, Cambridge (MA) and London (England)

Leung Y (1994) Inference with spatial knowledge: An artificial neural network approach *Geographical Systems* 1(2), 103-121

Leung Y (1997) Feedforward neural network models for spatial pattern classification In: Fischer, M M and Getis, A (eds) *Recent Developments in Spatial Analysis*: *Spatial Statistics, Behavioural Modelling and Computational Intelligence*, 336-59, Springer, Berlin

Leung Y, Leung K S, Ng W, Lau M I (1995) Evolving multilayer feedforward neural networks by genetic algorithms (unpublished paper)

Maniezzo V (1994) Genetic evolution of the topology and weight distribution of neural networks *IEEE Transactions on Neural Networks* 5(1), 39-53

Michalewicz Z (1992) *Genetic Algorithms + Data Structures = Evolution Programs* Springer, New York

Miller G F, Todd P M, Hedge S U (1989) Designing neural networks using genetic algorithms In: Schaffer, J D (ed) *Proceedings of the Third International Conference on Genetic Algorithms and their Applications* pp. 379-84 Morgan Kaufmann, San Mateo (CA)

Montana D J, Davis L (1989) Training feedforward networks using genetic algorithms In: Sridhara, N S (ed) *Eleventh International Joint Conference on Artificial Intelligence*, pp 762-7 Morgan Kaufmann, San Mateo (CA)

Moody J (1992) Generalization, weight decay and architecture selection for non-linear learning systems In: Moody J, Hanson J, Lippmann R (eds) *Advances in Neural Information Processing Systems IV*, 471-9 Morgan Kaufmann, San Mateo (CA)

Nijkamp P, Reggiani A, Tritapepe T (1996) Modelling an inter-urban flows in Italy: a comparison between neural network approach and logit analysis *Transportation Research C* 4(6), 323-38

Openshaw S (1988) Building an automated modelling system to explore a universe of spatial interaction models *Geographical Analysis* 20(1), 31-46

Openshaw S (1993) Modelling spatial interaction using neural net In: Fischer M M, Nijakmp P (eds) *Geographical Information Systems, Spatial Modelling, and Policy Evaluation*, pp. 147-64, Springer, Berlin

Openshaw S (1997) Neural network, genetic, and fuzzy logic models of spatial interaction, Paper submitted to *Envirnoment and Planning A*

Openshaw S, Openshaw C (1997) *Artificial intelligence in geography* Wiley, Chichester New York

Qi X, Palmieri F (1994) Theoretical analysis of evolutionary algorithms with an infinite population size in continuous space Part I: Basic properties of selection and mutation *IEEE Transactions on Neural Networks* 5(1), 102-19

Radcliffe N J (1991) Genetic set recombination and its application to neural network topology optimisation, EPCC Technical Report EPCC-TR-91-21, Edinburgh Parallel Computing Centre, University of Edinburgh

Rumelhart D E, Hinton G E, Williams R J (1986) Learning internal representations by error propagation In: Rumelhart D E, McClelland J L, The PDP Research Group (eds) *Parallel Distributed Processing: Explorations in the Microstructure of Cognitions, Volume 1: Foundations*, pp. 318-62 The MIT Press, Cambridge (MA)

Schaffer J D, Caruana, R A, Eshelman L J, Das R (1989) A study of control parameters affecting online performance of genetic algorithms for function optimization In: Schaffer J D (ed.) *Proceedings of the Third International Conference on Genetic Algorithms and their Applications*, pp. 51-60 Morgan Kaufmann, San Mateo (CA)

Spears M M, De Jong K A (1991) An analysis of multi-point crossover In: Rawlins G J E (ed) *Foundations of genetic algorithms*, pp. 301-315 Morgan Kaufmann, San Mateo (CA)

Syswerda G (1989) Uniform crossover in genetic algorithms In: Schaffer J D (ed) *Proceedings of the Third International Conference on genetic algorithms*, pp. 2-8 Morgan Kaufmann, San Mateo (CA)

White H (1989) Learning in artificial neural networks: A statistical perspective *Neural Computation* 1, 425-64

Whitley D, Hanson T (1989) Optimizing neural networks using faster, more accurate genetic search In: Schaffer, J D (ed) *Proceedings of the Third International Conference on Genetic Algorithms (Arlington, 1989),* 391-6 Morgan Kaufmann, San Mateo (CA)

Yao X (1993) A review of evolutionary artificial neural networks *International Journal of Intelligent Systems* 8(4), 539-67